PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 :
G06F 7/38, 7/42, 7/50

A1

(11) International Publication Number: WO 97/33223

(43) International Publication Date: 12 September 1997 (12.09.97)

(21) International Application Number: PCT/US97/03445

(22) International Filing Date: 4 March 1997 (04.03.97)

(30) Priority Data:
08/611,123 4 March 1996 (04.03.96) US

(71) Applicant *(for all designated States except US)*: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

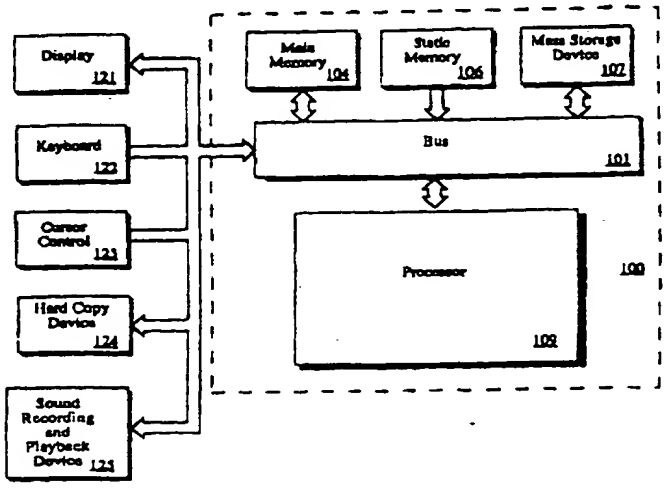
(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: LIN, Derrick, Chu [US/US]; 113 Barkentine Street, Foster City, CA 94404 (US). MOHEBBI, Mehrdad [IR/US]; 4251 Norwalk Drive AA305, San Jose, CA 95129 (US).

(74) Agents: TAYLOR, Edwin, H. et al.; Blakely, Sokoloff, Taylor & Zafman L.L.P., 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States: AL, AM, AT, AT (Utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), EE, EE (Utility model), ES, FI, FI (Utility model), GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).

Published
With international search report.

(54) Title: PACKED-ADD AND PACKED SUBTRACT OPERATIONS



(57) Abstract

A processor (109) having a circuit (203) for performing a packed addition and subtraction. The decoder (202) accesses the registers addressed by SRC1 and SRC2. These registers provide a first packed data and a second packed data to the circuit (203). Packed data consists of a number of fixed length data elements. The data elements can be eight, sixteen, or thirty-two bits in length. The circuit (203) performs the operation on the first data element from the first packed data and the first data element from the second packed data, producing a first result data element. The circuit (203) performs this operation on the next data element from the first packed data and the next data element from the second packed data, producing a next result data element. This continues for all the data elements in the first and second packed data. The result date elements constitute a result packed data that is stored in the destination register.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AM	Armenia	GB	United Kingdom	MW	Malawi
AT	Austria	GE	Georgia	MX	Mexico
AU	Australia	GN	Guinea	NE	Niger
BB	Barbados	GR	Greece	NL	Netherlands
BE	Belgium	HU	Hungary	NO	Norway
BF	Burkina Faso	IE	Ireland	NZ	New Zealand
BG	Bulgaria	IT	Italy	PL	Poland
BJ	Benin	JP	Japan	PT	Portugal
BR	Brazil	KE	Kenya	RO	Romania
BY	Belarus	KG	Kyrgystan	RU	Russian Federation
CA	Canada	KP	Democratic People's Republic of Korea	SD	Sudan
CF	Central African Republic			SE	Sweden
CG	Congo	KR	Republic of Korea	SG	Singapore
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LR	Liberia	SZ	Swaziland
CS	Czechoslovakia	LT	Lithuania	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	LV	Latvia	TJ	Tajikistan
DK	Denmark	MC	Monaco	TT	Trinidad and Tobago
EE	Estonia	MD	Republic of Moldova	UA	Ukraine
ES	Spain	MG	Madagascar	UG	Uganda
FI	Finland	ML	Mali	US	United States of America
FR	France	MN	Mongolia	UZ	Uzbekistan
GA	Gabon	MR	Mauritania	VN	Viet Nam

PACKED-ADD AND AND PACKED SUBTRACT OPERATIONS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of processor operation. In particular, the present invention describes arithmetic operations using a control signal to manipulate multiple data elements. The present invention allows execution of addition and subtraction operations on packed data types.

2. Description of Related Art

Today, most personal computer systems operate with one instruction to produce one result. Performance increases are achieved by increasing execution speed of instructions and the processor instruction complexity; known as Complex Instruction Set Computer (CISC). Such processors as the Intel 80286™ microprocessor, available from Intel Corp. of Santa Clara, California, belong to the CISC category of processor.

Previous computer system architecture has been optimized to take advantage of the CISC concept. Such systems typically have data buses thirty-two bits wide. However, applications targeted at computer supported conferencing (CSC - the integration of teleconferencing with mixed media data manipulation), 2D/3D graphics, image processing, video compression/decompression, recognition algorithms and audio manipulation increase the need for improved performance. But, increasing the execution speed and complexity of instructions is only one solution.

One common aspect of these applications is that they often manipulate large amounts of data where only a few bits are important. That is, data whose relevant bits are represented in much fewer bits than the size of the data bus. For example, processors execute many operations on eight bit and

sixteen bit data (e.g., pixel color components in a video image) but have much wider data busses and registers. Thus, a processor having a thirty-two bit data bus and registers, and executing one of these algorithms, can waste up to seventy-five percent of its data processing, carrying and storage capacity because only the first eight bits of data are important.

As such, what is desired is a processor that increases performance by more efficiently using the difference between the number of bits required to represent the data to be manipulated and the actual data carrying and storage capacity of the processor.

SUMMARY OF THE INVENTION

A microprocessor for performing packed arithmetic operations on two packed data. The apparatus includes a first bus coupled to portions of first inputs of an adder that correspond to the smallest values representable by the packed data. The apparatus includes a second bus coupled to portions of second inputs of the adder that correspond to the smallest values representable by the packed data. The apparatus also includes at least one mux coupled to at least one portion between portions of first inputs and at least one mux coupled to at least one portion between portions of second inputs. Each of the muxes control whether a carry is input to the higher order adjacent value. A packed result of said packed arithmetic operation is indicated on a portion of said plurality of outputs representing multiple values.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates an embodiment of the computer system capable of implementing the present invention.

Figure 2 illustrates an embodiment of the processor of the present invention.

Figure 3 is a flow diagram illustrating the general steps used by the processor to manipulate data in the register file.

Figure 4a illustrates memory data types.

Figure 4b through Figure 4d illustrate in-register integer data representations.

Figure 5a illustrates packed data-types.

Figure 5b through Figure 5d illustrate in-register packed data representations.

Figure 6 illustrates the control signal format used in the computer system to indicate the use of packed data.

Figure 7a illustrates the steps followed by the processor when adding packed data to packed data.

Figure 7b illustrates the steps followed by the processor when subtracting packed data from packed data.

Figure 8 illustrates a circuit capable of manipulating individual bits of packed data.

Figure 9 illustrates one embodiment of a circuit implementing the present invention whereby the circuit is capable of manipulating packed byte data.

Figure 10 illustrates a logical view of one embodiment of the present invention whereby the circuit is capable of manipulating packed word data.

Figure 11 illustrates a logical view of an alternate embodiment of the present invention whereby the circuit is capable of manipulating packed doubleword data.

Figure 12 illustrates another embodiment of a Packed Add/Subtract circuit.

Figure 13 illustrates one embodiment of a Logic 1200.

Figure 14 illustrates one embodiment of a Logic 1201.

Figure 15 is a flow graph describing one embodiment of the steps for performing a packed add operation.

Figure 16 is a flow graph describing one embodiment of the steps for performing a packed subtract operation.

Figure 17 is a flow graph describing one embodiment of the steps for performing a packed compare greater-than operation

DEFINITIONS

To provide a foundation for understanding the description of the embodiments of the present invention, the following definitions are provided.

Bit X through Bit Y:
defines a subfield of binary number. For example, bit six through bit zero of the byte 00111010_2 (shown in base two) represent the subfield 1110102. The '2' following a binary number indicates base 2. Therefore, 10002 equals 8_{10}, while F_{16} equals 15_{10}.

R_X: is a register. A register is any device capable of storing and providing data. Further functionality of a register is described below.

DEST: is an address of a register.
SRC1: is an address of a register.
SRC2: is an address of a register.

Result: is the data to be stored in the register addressed by DEST.
Source1: is the data stored in the register addressed by SRC1.
Source2: is the data stored in the register addressed by SRC2.

COMPUTER SYSTEM

An apparatus and method of increasing processor performance by efficiently reducing the difference between data element length and actual carrying and storage capacity is described. In the following description,

numerous specific details are set forth such as packed data formats, control signal formats, etc., in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known processor architecture, circuits, structures and techniques have not been shown in detail in order not to unnecessarily obscure the present invention.

Figure 1 illustrates a computer system 100 upon which one embodiment of the present invention can be implemented. Computer system 100 comprises a bus or other communication means 101 for communicating information, and a processing means 109 coupled with bus 101 for processing information. System 100 further comprises a random access memory (RAM) or other dynamic storage device 104 (referred to as main memory), coupled to bus 101 for storing information and instructions to be executed by processor 109. Main memory 104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 109. Computer system 100 also comprises a read only memory (ROM) and/or other static storage device 106 coupled to bus 101 for storing static data and instructions for processor 109. Data storage device 107 is coupled to bus 101 for storing information and instructions. Memory includes any data storage medium, such as main memory 104, cache memory, registers, ROM, and other static storage devices.

Furthermore, a data storage device 107 such as a magnetic disk or optical disk and its corresponding disk drive can be coupled to computer system 100. Computer system 100 can also be coupled via bus 101 to a display device 121, such as a cathode ray tube (CRT), for displaying information to a computer user. An alphanumeric input device 122, including alphanumeric and other keys, is typically coupled to bus 101 for communicating information and command selections to processor 109. Another type of user input device is cursor control 123, such as a mouse, a trackball, or cursor direction keys for communicating direction information

and command selections to processor 109 and for controlling cursor movement on display 121. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane.

Another device that may be coupled to bus 101 is hard copy device 124 that may be used for printing instructions, data, or other information on a medium such as paper, film, or similar types of media. Additionally, computer system 100 can be coupled to a device for sound recording and/or playback 125 such as an audio digitizer coupled to a microphone for recording information. Further, the device may include a speaker that is coupled to a digital to analog (D/A) converter for playing back the digitized sounds. Finally, computer system 100 can be a terminal in a computer network (e.g., a LAN).

Computer system 100 is useful for supporting computer supported conferencing (CSC - the integration of teleconferencing with mixed media data manipulation), 2D/3D graphics, image processing, video compression/decompression, recognition algorithms and audio manipulation.

PROCESSOR

Figure 2 illustrates a detailed diagram of processor 109. One skilled in the art would understand that processor 109 could be implemented on one or more substrates using any of a number of process technologies, such as, BiCMOS, CMOS, and NMOS.

Processor 109 comprises a decoder 202 for decoding control signals and data used by processor 109. Data can then be stored in register file 204 via internal bus 205. As a matter of clarity, register file 204 should not be limited in meaning to a particular type of circuit. Rather, register file 204 need only be capable of storing and providing data, and performing the functions described herein.

Depending on the type of data, the data may be stored in integer registers 209, status registers 208, or instruction pointer register 211. Integer

registers 209 contains thirty-two integer registers, R_1 212a through R_n 212af. Each register is sixty-four bits in length. R_1 212a, R_2 212b and R_3 212c are examples of individual registers in integer registers 209. Status registers 208 indicate the status of processor 109. Instruction pointer 211 stores the address of the next instruction to be executed. Integer registers 209, status registers 208, and instruction pointer register 211 all connect to internal bus 205.

Arithmetic logic unit 203 (ALU) performs the arithmetic and logic operations carried out by processor 109. Such operations may include logical shifts, addition, subtraction and multiplication, etc. ALU 203 connects to internal bus 205. Cache 206 is an optional element of processor 109 and can be used to cache data, including control signals, from, for example, main memory 104. Cache 206 is connected to decoder 202, and is connected to receive control signal 207.

Figure 3 illustrates the general operation of processor 109. That is, Figure 3 illustrates the steps followed by processor 109 while performing an operation on packed data, performing an operation on unpacked data, or performing some other operation. For example, such operations include a load operation to load a register in register file 204 with data from cache 206, main memory 104, static memory 106, or mass storage device 107. In one embodiment of the present invention, processor 109 supports most of the instructions supported by the Intel 80486™, available from Intel Corporation of Santa Clara, California. In another embodiment of the present invention, processor 109 supports all the operations supported by the Intel 80486™, available from Intel Corporation of Santa Clara, California. In another embodiment of the present invention, processor 109 supports all the operations supported by the Pentium™, the Intel 80486™, the 80386™, the Intel 80286™, and the Intel 8086™, all available from Intel Corporation of Santa Clara, California. In another embodiment of the present invention, processor 109 supports all the operations supported in the X86 Architecture™ as defined by Intel Corporation of Santa Clara, California (see

Microprocessors, Intel Data Books volume 1 and volume 2, 1992 and 1993, available from Intel of Santa Clara, California) .

At block 301, the decoder 202 receives a control signal 207 from either the cache 206 or bus 101. Decoder 202 decodes the control signal to determine the operations to be performed.

Decoder 202 accesses the register file 204, or a location in another memory, at block 302. Registers in the register file 204, or memory locations in another memory, are accessed depending on the address specified in the control signal 207. For example, for an operation on packed data, control signal 207 includes SRC1, SRC2 and DEST register addresses. SRC1 is the address of the first source register. SRC2 is the address of the second source register. In some cases, the SRC2 address is optional as not all operations require two source addresses. If the SRC2 address is not required for an operation, then only the SRC1 address is used. DEST is the address of the destination register where the result data is stored. SRC1, SRC2 and DEST are described more fully in relation to Figure 6. The data stored in these registers is referred to as Source1, Source2, and Result respectively. Each of these data is sixty-four bits in length.

In another embodiment of the present invention, any one, or all, of SRC1, SRC2 and DEST, can define a memory location in the addressable memory space of processor 109. For example, SRC1 may identify a memory location in main memory 104 while SRC2 identifies a first register in integer registers 201, and DEST identifies a second register in registers 209. For simplicity of the description herein, references are made to the accesses to the register file 204, however, these accesses could be made to another memory instead.

In another embodiment of the present invention, the operation code only includes two addresses, SRC1 and SRC2. In this embodiment, the result of the operation is stored in the SRC1 or SRC2 register. That is SRC1 (or SRC2) is used as the DEST. This type of addressing is compatible with previous CISC instructions having only two addresses. This reduces the

complexity in the decoder 202. Note, in this embodiment, if the data contained in the SRC1 register is not to be destroyed, then that data is copied into another register before the execution of the operation. The copying would require an additional instruction. To simplify the description herein, the three address addressing scheme will be described (i.e. SRC1, SRC2, and DEST). However, it should be remembered that the control signal, in one embodiment, may only include SRC1 and SRC2, and that SRC1 (or SRC2) identifies the destination register.

Where the control signal requires an arithmetic or logic operation, at step 303, ALU 203 will be enabled to perform this operation on accessed data from register file 204. Once the arithmetic or logic operation has been performed in ALU 203, at step 304, the result is stored back into register file 204 according to requirements of control signal 207.

DATA AND STORAGE FORMATS

Figure 4a illustrates some of the data formats as may be used in the computer system of figure 1. Processor 109 can manipulate these data formats. Multimedia algorithms often use these data formats. A byte 401 contains eight bits of information. A word 402 contains sixteen bits of information, or two bytes. A doubleword 403 contains thirty-two bits of information, or four bytes. Thus, processor 109 executes control signals that may operate on any one of these memory data formats.

In the following description, references to bit, byte, word, and doubleword subfields are made. For example, bit six through bit zero of the byte 00111010_2 (shown in base 2) represent the subfield 111010_2.

Figure 4b through Figure 4d illustrate in-register integer data representations used in one embodiment of the present invention. For example, unsigned byte 410 can represent data stored in integer register R_1 212a. A register, in integer registers 209, is sixty-four bits in length.

In-register unsigned byte representation 410 illustrates processor 109 storing a byte 401 in integer registers 209, the first eight bits, bit seven

through bit zero, in that integer register are dedicated to the data byte 401. These bits are shown as {b}. To properly represent this byte, the remaining 56 bits must be zero.. For a signed byte 411, integer registers 209 store the data in the first seven bits, bit six through bit zero, to be data. The seventh bit represents the sign bit, shown as an {s}. The remaining bit sixty-three through bit eight are the continuation of the sign for the byte.

Unsigned word in-register data representation 412 is stored in one register of integer registers 209. Bit fifteen through bit zero contain the unsigned word 402. These bits are shown as {w}. To properly represent this word, the remaining bit sixty-three through bit sixteen must be zero. Signed word 402 is stored in bit fourteen through bit zero as shown in the signed word in-register data representation 413. The remaining bit sixty-three through bit fifteen is the sign field.

A doubleword 403 can be stored as an unsigned doubleword or a signed doubled word 414 or 415. Bit thirty-one through bit zero of an unsigned doubleword 414 are the data. These bits are shown as {d}. To properly represent this unsigned doubleword, the remaining bit sixty-three through bit thirty-two must be zero. Integer register 209 stores a signed doubleword 415 in its bit thirty through bit zero; the remaining bit sixty-three through bit thirty-one are the sign field.

As indicated by the above Figure 4b through Figure 4d, storage of some data types in a sixty-four bit wide register is an inefficient method of storage. For example, for storage of a byte in-register data representation 410 bit sixty-three through bit eight must be zero, while only bit seven through bit zero may contain non-zero bits. Thus, a processor storing a byte in an integer register uses only 12.5% of its capacity. Similarly, only the first few bits of operations performed by ALU 203 will be important.

Figure 5a illustrates the data formats for packed data. Three packed data formats are illustrated; packed byte 501, packed word 502, and packed doubleword 503. Packed byte, in the preferred embodiment of the present invention, is sixty-four bits long containing eight data elements. Each data

element is one byte long. Generally, a data element is an individual piece of data that is stored in a single integer register with other data elements of the same length. In the preferred embodiment of the present invention, the number of data elements stored in an integer register is sixty-four bits divided by the length in bits of a data element.

Packed word data 502 is sixty-four bits long and contains four word data elements 402. Each word data element 402 contains sixteen bits of information.

Packed doubleword data 503 is sixty-four bits long and contains two doubleword 403 data elements. Each doubleword data element 403 contains thirty-two bits of information.

Figure 5b through Figure 5d illustrate the in-register packed data storage representation. Unsigned packed byte in-register representation 510 illustrates the storage of packed data type 501 in one of the integer registers R_0 212a through R_n 212af. Information for each byte data element is stored in bit seven through bit zero for byte zero, bit fifteen through bit eight for byte one, bit twenty-three through bit sixteen for byte two, bit thirty-one through bit twenty-four for byte three, bit thirty-nine through bit thirty-two for byte four, bit forty-seven through bit forty for byte five, bit fifty-five through bit forty-eight for byte six and bit sixty-three through bit fifty-six for byte seven. Thus, all available bits are used in the integer register. This storage arrangement increases the storage efficiency of the processor. As well, with eight data elements accessed, one operation can now be performed on eight data elements simultaneously. Signed packed byte in-register representation 511 is similarly stored in an integer register in integer registers 209. Note that only the eighth bit of every byte data element is the sign bit; no other bits are used to indicate sign.

Unsigned packed word in-register representation 512 illustrates how word three through word zero are stored in one integer register of integer registers 209. Bit fifteen through bit zero contain the data element information for word zero, bit thirty-one through bit sixteen contain the information for

data element word one, bit forty-seven through bit thirty-two contain the information for data element word two and bit sixty-three through bit forty-eight contain the information for data element word three. Signed packed word in-register representation 513 is similar to the unsigned packed word in-register representation 512. Note that only the sixteenth bit of each word data element contains the sign indicator.

Unsigned packed doubleword in-register representation 514 shows how integer registers 209 store two doubleword data elements. Doubleword zero is stored in bit thirty-one through bit zero of the integer register. Doubleword one is stored in bit sixty-three through bit thirty-two of the integer register. Signed packed doubleword in-register representation 515 is similar to unsigned packed doubleword in-register representation 514. Note that the signed bit is the thirty-second bit of the doubleword data element.

In one embodiment of the present invention, the individual programming processor 109 must track whether an addressed register, R_1 212a for example, is storing packed data or simple integer data. One skilled in the art would understand that in an alternative embodiment, processor 109 could track the type of data stored in individual registers of integer registers 209. This alternative embodiment could then generate errors if, for example, a packed addition operation were attempted on simple integer data.

CONTROL SIGNAL FORMATS

The following describes the control signal formats used by processor 109 to manipulate packed data. In one embodiment of the present invention, control signals are represented as thirty-two bits. Decoder 202 may receive control signal 207 from bus 101. However, it is obvious to one skilled in the art that decoder 202 can also receive such control signals from cache 206.

Figure 6 illustrates the general format for a control signal operating on packed data. Operation field OP 601, bit thirty-one through bit twenty-six, provides information about the operation to be performed by processor 109; for example, packed addition, packed subtraction, etc.. SRC1 602, bit

twenty-five through twenty, provides the source register address of a register in integer registers 209. This source register contains the first packed data, Source1, to be used in the execution of the control signal. Similarly, SRC2 603, bit nineteen through bit fourteen, contains the address of a register in integer registers 209. This second source register contains the packed data, Source2, to be used during execution of the operation. DEST 605, bit five through bit zero, contains the address of a register in integer registers 209. This destination register will store the result packed data, Result, of the packed data operation. Although this general format for a control signal uses source and destination registers in integer registers 209, one skilled in the art would understand that an alternative embodiment can use the source and destination addresses of memory locations in cache 206 or main memory 104.

Control bits SZ 610, bit twelve and bit thirteen, indicates the length of the data elements in the first and second packed data source registers. If SZ 610 equals 01_2, then the packed data is formatted as packed byte data type 501. If SZ 610 equals 10_2, then the packed data is formatted as packed word data type 502. SZ 610 equaling 00_2 or 11_2 is reserved, however, given the above description, one of ordinary skill in the art would understand that one of these values could be used for doubleword size data type 503.

Control bit T 611, bit eleven, indicates whether the operation is to be carried out with saturate mode. If T 611 equals one, then a saturating operation is performed. If T 611 equals zero, then a nonsaturating operation is performed. Saturating operations will be described later.

Control bit S 612, bit ten, indicates the use of a signed operation. If S 612 equals one, then a signed operation is performed. If S 612 equals zero, then an unsigned operation is performed.

DESCRIPTION OF SATURATE/UNSATURATE

As mentioned previously, T 611 indicates whether operations optionally saturate. Where the result of an operation, with saturate enabled, overflows or underflows the range of the data, the result will be clamped. Clamping means setting the result to a maximum or minimum value should a

result exceed the range's maximum or minimum value. In the case of underflow, saturation clamps the result to the lowest value in the range and in the case of overflow, to the highest value. The allowable range for each data format is shown in Table 1.

Data Format	Minimum Value	Maximum Value
Unsigned Byte	0	255
Signed Byte	-128	127
Unsigned Word	0	65535
Signed Word	-32768	32767
UnSigned Doubleword	0	$2^{64}-1$
Signed Doubleword	-2^{63}	$2^{63}-1$

Table 1

As mentioned above, T 611 indicates whether saturating operations are being performed. Therefore, using the unsigned byte data format, if an operation's result = 258 and saturation was enabled, then the result would be clamped to 255 before being stored into the operation's destination register. Similarly, if an operation's result = -32999 and processor 209 used signed word data format with saturation enabled, then the result would be clamped to -32768 before being stored into the operation's destination register.

PACKED DATA ARITHMETIC

The preferred embodiment of the present invention increases the performance of multimedia algorithms by supporting packed data arithmetic operations in addition to unpacked data arithmetic operations. In particular, it has been found that packed addition and subtraction increase the performance of these algorithms significantly. Packed data operations execute independently on all eight (packed bytes), four (packed words), or, in an alternative embodiment, two (packed doublewords) data elements.

PACKED ADDITION

The preferred embodiment of the present invention enables packed addition operations to be performed in ALU 203. That is, the present invention enables each data element of a first packed data to be added individually to each data element of a second packed data. Figure 7a illustrates one method of performing a packed addition in the present invention. At step 701, decoder 202 decodes control signal 207 received by processor 109. Thus, decoder 202 decodes: the operation code for packed addition; SRC1 602, SRC2 603 and DEST 605 addresses in integer registers 209; saturate/unsaturate, signed/unsigned, and length of the data elements in the packed data. At step 702, via internal bus 205, decoder 202 accesses integer registers 209 in register file 204 given the SRC1 602 and SRC2 603 addresses. Integer registers 209 provides ALU 203 with the packed data stored in the registers at these addresses, Source1 and Source2 respectively. That is, integer registers 209 communicate the packed data to ALU 203 via internal bus 205.

At step 703, decoder 202 enables ALU 203 to perform a packed addition operation. Decoder 202 further communicates, via internal bus 205, the length of packed data elements, whether saturation is to be used, and whether signed arithmetic is to be used. At step 704, the length of the data element determines which step is to be executed next. If the length of the data elements in the packed data is eight bits (byte data), then ALU 203 performs step 705a. However, if the length of the data elements in the packed data is sixteen bits (word data), then ALU 203 performs step 705b. In the present embodiment of the invention, only eight bit and sixteen bit data element length packed addition is supported. However, in alternative embodiments of the present invention, thirty-two bit data element length packed addition is also supported.

Assuming the length of the data elements is eight bits, then step 705a is executed. ALU 203 adds bit seven through bit zero of Source1 to bit seven through bit zero of SRC2, producing bit seven through bit zero of Result

packed data. In parallel with this addition, ALU 203 adds bit fifteen through bit eight of Source1 to bit fifteen through bit eight of Source2, producing bit fifteen through bit eight of Result packed data. In parallel with these additions, ALU 203 adds bit twenty-three through bit sixteen of Source1 to bit twenty-three through bit sixteen of Source2, producing bit twenty-three through bit sixteen of Result packed data. In parallel with these additions, ALU 203 adds bit thirty-one through bit twenty-four of Source1 to bit thirty-one through bit twenty-four of Source2, producing bit thirty-one through bit twenty-four of Result packed data. In parallel with these additions, ALU 203 adds bit thirty-nine through bit thirty-two of Source1 to bit thirty-nine through bit thirty-two of Source2, producing bit thirty-nine through bit thirty-two of Result packed data. In parallel with these additions, ALU 203 adds bit forty-seven through bit forty of Source1 to bit forty-seven through bit forty of Source2, producing bit forty-seven through bit forty of Result packed data. In parallel with these additions, ALU 203 adds bit fifty-five through bit forty-eight of Source1 to bit fifty-five through bit forty-eight of Source2, producing bit fifty-five through bit forty-eight of Result packed data. In parallel with these additions, ALU 203 adds bit sixty-three through bit fifty-six of Source1 to bit sixty-three through bit fifty-six of Source2, producing bit sixty-three through bit fifty-six of Result packed data.

Assuming the length of the data elements is sixteen bits, then step 705b is executed. ALU 203 adds bit fifteen through bit zero of Source1 to bit fifteen through bit zero of SRC2, producing bit fifteen through bit zero of Result packed data. In parallel with this addition, ALU 203 adds bit thirty-one through bit sixteen of Source1 to bit thirty-one through bit sixteen of Source2, producing bit thirty-one through bit sixteen of Result packed data. In parallel with these additions, ALU 203 adds bit forty-seven through bit thirty-two of Source1 to bit forty-seven through bit thirty-two of Source2, producing bit forty-seven through bit thirty-two of Result packed data. In parallel with these additions, ALU 203 adds bit sixty-three through bit forty-eight of Source1 to

bit sixty-three through bit forty-eight of Source2, producing bit sixty-three through bit forty-eight of Result packed data.

At step 706, decoder 202 enables a register in integer registers 209 with DEST 605 address of the destination register. Thus, the Result is stored in the register addressed by DEST 605.

Table 2a illustrates the in-register representation of packed addition operation. The first row of bits is the packed data representation of a Source1 packed data. The second row of bits is the packed data representation of a Source2 packed data. The third row of bits is the packed data representation of the Result packed data. The number below each data element bit is the data element number. For example, Source1 data element 0 is 10001000_2. Therefore, if the data elements are eight bits in length (byte data), and unsigned, unsaturated addition is performed, the ALU 203 produces the Result packed data as shown.

Note that in one embodiment of the present invention, where a result overflows or underflows and the operation is using unsaturate, that result is simply truncated. That is, the carry bit is ignored. For example, in Table 2a, the in-register representation of result data element one would be: $10001000_2 + 10001000_2 = 00001000_2$. Similarly, for underflows, the result is truncated. This form of truncation enables a programmer to easily perform modulo arithmetic. For example, an equation for result data element one can be expressed as: (Source1 data element one + Source2 data element one) mod 256 = result data element one. Further, one skilled in the art would understand from this description that overflows and underflows could be detected by setting error bits in a status register.

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
+ 7	+ 6	+ 5	+ 4	+ 3	+ 2	+ 1	+ 0
10101010	01010101	10101010	10000001	10000000	11110000	11001111	10001000
= 7	= 6	= 5	= 4	= 3	= 2	= 1	= 0
11010100	10101010	11111111	Overflow	Overflow	Overflow	Overflow	Overflow
7	6	5	4	3	2	1	0

Table 2a

Table 2b illustrates the in-register representation of a packed word data addition operation. Therefore, if the data elements are sixteen bits in length (word data), and unsigned, unsaturated addition is performed, the ALU 203 produces the Result packed data as shown. Note that in word data element two, the carry from bit seven (see emphasized bits *1* below) propagated into bit eight, causing data element two to overflow (see emphasized ***overflow*** below).

00101010 01010101	01010101 *1*1111111	10000000 01110000	10001111 10001000
+ 3	+ 2	+ 1	+ 0
10101010 01010101	10101010 *1*0000001	10000000 11110000	11001111 10001000
= 3	= 2	= 1	= 0
11010100 10101010	***Overflow***	Overflow	Overflow
3	2	1	0

Table 2b

Table 2c illustrates the in-register representation of packed doubleword data addition operation. This operation is supported in an alternative embodiment of the present invention. Therefore, if the data elements are thirty-two bits in length (i.e., doubleword data), and unsigned, unsaturated addition is performed, the ALU 203 produces the Result packed

data as shown. Note that carries from bit seven and bit fifteen of doubleword data element one propagated into bit eight and bit sixteen respectively.

00101010 01010101 01010101 11111111	10000000 01110000 10001111 10001000
1 +	2 +
10101010 01010101 10101010 10000001	10000000 11110000 11001111 10001000
1 =	2 =
11010100 10101011 00000000 10000000	Overflow
1	0

Table 2c

To better illustrate the difference between packed addition and ordinary addition, the data from the above example is duplicated in Table 3. However, in this case, ordinary addition (sixty-four bit) is performed on the data. Note that the carries from bit seven, bit fifteen, bit twenty-three, bit thirty-one, bit thirty-nine and bit forty-seven have been carried into bit eight, bit sixteen, bit twenty-four, bit thirty-two, bit forty and bit forty-eight respectively.

00101010	01010101	01010101	11111111	10000000	01110000	10001111	10001000
±							
10101010	01010101	10101010	10000001	10000000	11110000	11001111	10001000
≡							
11010100	10101011	00000000	10000001	00000001	01100001	01011111	00010000

Table 3

SIGNED/UNSATURATE PACKED ADDITION

Table 4 illustrates an example of a signed packed addition where the data element length of the packed data is eight bits. Saturation is not used. Therefore, results can overflow and underflow. Table 4 uses different data than Tables 1a–1c and Table 3.

00101010	01010101	01010101	01111111	00000000	11110000	00001111	10001000
+	+	+	+	+	+	+	+
10101010	01010101	10101010	00000001	00000000	11110000	00001111	10001000
=	=	=	=	=	=	=	=
11010100	Overflow	11111111	Overflow	00000000	Underflow	00011110	Underflow
7	6	5	4	3	2	1	0

Table 4

SIGNED/SATURATE PACKED ADDITION

Table 5 illustrates an example of a signed packed addition where the data element length of the packed data is eight bits. Saturate is used, therefore, overflow will be clamped to the maximum value, and underflow will be clamped to the minimum value. Table 5 uses the same data as Table 4. Here data element zero and data element two are clamped to the minimum value, while data element four and data element six are clamped to the maximum value.

00101010	01010101	01010101	01111111	00000000	11110000	00001111	10001000
+ 7	+ 6	+ 5	+ 4	+ 3	+ 2	+ 1	+ 0
10101010	01010101	10101010	00000001	00000000	11110000	00001111	10001000
= 7	= 6	= 5	= 4	= 3	= 2	= 1	= 0
11010100	01111111	11111111	01111111	00000000	10000000	00011110	10000000
7	6	5	4	3	2	1	0

Table 5

PACKED SUBTRACTION

The preferred embodiment of the present invention enables packed subtraction operations to be performed in ALU 203. That is, the present invention enables each data element of a second packed data to be subtracted individually from each data element of a first packed data. Figure 7b illustrates one method of performing a packed subtraction in the present invention. Note that steps 710-713 are similar to steps 701-704.

In the present embodiment of the invention, only eight bit and sixteen bit data element length packed subtraction is supported. However, in alternative embodiments of the present invention, thirty-two bit data element length packed subtraction is also supported. Further, although in the present embodiment of the invention, subtraction is performed by subtracting Source2 from Source1, in an alternate embodiment, subtraction is performed by subtracting Source1 from Source2.

Assuming data element length is eight bits, steps 714a and 715a are executed. ALU 203 2's complements bit seven through bit zero of Source2. In parallel with this 2's complement, ALU 203 2's complements bit fifteen through bit eight of Source2. In parallel with these 2's complements, ALU 203 2's complements bit twenty-three through bit sixteen of Source2. In parallel with these 2's complements, ALU 203 2's complements bit thirty-one through bit twenty-four of Source2. In parallel with these 2's complements, ALU 203 2's complements bit thirty-nine through bit thirty-two of Source2. In parallel with these 2's complements, ALU 203 2's complements bit forty-seven through bit forty of Source2. In parallel with these 2's complements, ALU 203 2's complements bit fifty-five through bit forty-eight of Source2. In parallel with these 2's complements, ALU 203 2's complements bit sixty-three through bit fifty-six of Source2. At step 715a, ALU 203 performs the addition of the 2's complemented bits of Source2 to the bits of Source1 as generally described for step 705a.

Assuming data element length is sixteen bits, steps 714b and 715b are executed. ALU 203 2's complements bit fifteen through bit zero of Source2. In parallel with this 2's complement, ALU 203 2's complements bit thirty-one through bit sixteen of Source2. In parallel with these 2's complements, ALU 203 2's complements bit forty-seven through bit thirty-two of Source2. In parallel with these 2's complements, ALU 203 2's complements bit sixty-three through bit forty-eight of Source2. At step 715a, ALU 203 performs the addition of the 2's complemented bits of Source2 to the bits of Source1 as generally described for step 705b.

Note that steps 714 and 715 are the method used in one embodiment of the present invention to subtract a first number from a second number. However, other forms of subtraction are known in the art and this invention should not be considered limited to using 2's complement arithmetic.

At step 716, decoder 202 enables integer registers 209 with the destination address of the destination register. Thus, the result packed data is stored in the DEST register of integer registers 209.

Table 6 illustrates the in-register representation of packed subtraction operation. Assuming the data elements are eight bits in length (byte data), and unsigned, unsaturated subtraction is performed, then ALU 203 produces the result packed data as shown.

00101010	01010101	01010101	01111111	00000000	11110000	00001111	10001000
- 7	- 6	- 5	- 4	- 3	- 2	- 1	- 0
10101010	01010101	10101010	00000001	00000000	11110000	00001111	10001000
= 7	= 6	= 5	= 4	= 3	= 2	= 1	= 0
Underflow	00000000	Underflow	01111110	00000000	00000000	00000000	00000000
7	6	5	4	3	2	1	0

Table 6

PACKED DATA CIRCUITS

The preferred embodiment of the present invention employs numerous circuits to implement the invention. Figure 8 illustrates a circuit capable of manipulating individual bits of packed data. Figure 9 illustrates a view of one embodiment of the present invention whereby a circuit is capable of manipulating packed byte data. Figures 10 and 11 illustrate logical views of the Figure 9 circuit whereby the circuit is enabled to manipulate packed word and packed doubleword data respectively.

Figure 8 illustrates a modified bit slice adder/subtractor. Adder/subtractor 801a-b enable two bits from Source2 to be added to, or subtracted from, Source1. Operation and carry control 803 transmits to control 809a control signals to enable an addition or subtraction operation. Thus, adder/subtractor 801a adds or subtracts bit i received on Source2$_i$ 805a to bit i received on Source1$_i$ 804a, producing a result bit transmitted on Result$_i$ 806a. C_{in} 807a-b and C_{out} 808a-b represent carry control circuitry as is commonly found on adder/subtractors.

Bit control 802 is enabled from operation and carry control 803 via packed data enable 811 to control $C_{in_{i+1}}$ 807b and C_{out_i}. For example, in Table 7a, an unsigned packed byte addition is performed. If adder/subtractor 801a adds Source1 bit seven to Source2 bit seven, then operation and carry control 803 will enable bit control 802, stopping the propagation of a carry from bit seven to bit eight.

...	...	...	...	...	...	00001111	10001000
+ 7	+ 6	+ 5	+ 4	+ 3	+ 2	+ 1	+ 0
...	...	...	...	...	...	00001111	10001000
= 7	= 6	= 5	= 4	= 3	= 2	= 1	= 0
...	...	...	...	...	...	00011110	Overflow
7	6	5	4	3	2	1	0

Table 7a

However, if an unsigned packed word addition is performed, and 801a is similarly used to add bit seven of Source1 to bit seven of Source2, bit control 802 propagates the carry to bit eight. Table 7b illustrates this result. This propagation would be allowed for packed doubleword addition as well as unpacked addition.

...	...	...	00001111 10001000
+ 3	+ 2	+ 1	+ 0
...	...	...	00001111 10001000
= 3	= 2	= 1	= 0
...	...	...	00011111 00010000
3	2	1	0

Table 7b

Adder/subtractor 801a subtracts bit $Source2_i$ 805a from $Source1_i$ 804a by first forming the 2's complement of $Source2_i$ 805a by inverting $Source2_i$ 805a and adding one. Then adder/subtractor 801a adds this result to $Source1_i$

804a. Bit slice 2's complementing techniques are well known in the art, and one skilled in the art would understand how to design such a bit slice 2's complementing circuit. Note that propagation of carries are control by bit control 802 and operation and carry control 803.

Figure 9 illustrates a circuit capable of performing a packed byte addition on two sixty-four bit registers. Source1 bus 901 and Source2 bus 902 carry the information signals to the adder/subtractors 908a-h via $Source1_{in}$ 906a-h and $Source2_{in}$ 905a-h respectively. Thus, adder/subtractor 908a adds/subtracts Source2 bit seven through bit zero to/from Source1 bit seven through bit zero; adder/subtractor 908b adds/subtracts Source2 bit fifteen through bit eight to/from Source1 bit fifteen through bit eight, etc.. CTRL 904a-h receives, from ALU control 903, via packed control 911, control signals disabling the propagation of carries, enabling/disabling saturate, and enabling/disabling signed/unsigned arithmetic. ALU control 903 disables propagation of carries by receiving carry information from CTRL 904a-h and not propagating it to the next most significant adder/subtractor 908a-h. Thus, ALU control 903 performs the operations of the operation and carry control 803 and the bit control 802 for 64 bit packed data. One skilled in the art would be able create such a circuit given the illustrations in Figures 1-9 and the above description.

Adder/subtractors 908a-h communicate result information, via result out 907a-h, of the various packed additions to result register 910a-h. Each result register 910a-h stores and then transmits the result information onto Result bus 909. This result information is then stored in the integer register specified by the DEST 605 register address.

Figure 10 illustrates a logical view of the operation of the circuit of Figure 9. Here, packed word operations are being performed. Propagations of carries between bit eight and bit seven, bit twenty-four and bit twenty-three, bit forty and bit thirty-nine, and bit fifty-six and bit fifty-five are enabled by ALU control 903. Thus, adder/subtractor 908a and 908b, shown as virtual adder/subtractor 1008a, will act together to add/subtract the first word of

packed word data Source2 (bit fifteen through bit zero) to/from the first word of packed word data Source1 (bit fifteen through bit zero); adder/subtractor 908c and 908d, shown as virtual adder/subtractor 1008b, will act together to add/subtract the second word of packed word data Source2 (bit thirty-one through bit sixteen) to/from the second word of packed word data Source1 (bit thirty-one through bit sixteen), etc..

Virtual adder/subtractors 1008a-d communicate result information, via result out 1007a-d (combined result outs 907a-b, 907c-d, 907e-f and 907g-h), to virtual result registers 1010a-d. Each virtual result register 1010a-d (combined result registers 910a-b, 910c-d, 910e-f and 910g-h) stores a sixteen bit result data element to be communicated onto Result bus 909.

Although, the preferred embodiment of the present invention does not support packed doubleword addition and subtraction, an alternative embodiment would support these operations. Figure 11 illustrates a logical view of such an operation of the circuit of Figure 9. Propagations of carries between bit eight and bit seven, bit sixteen and bit fifteen, bit twenty-four and bit twenty-three, bit forty and bit thirty-nine, bit forty-eight and bit forty-seven, and bit fifty-six and bit fifty-five are enabled by ALU control 903. Thus, adder/subtractors 908a-d, shown as virtual adder/subtractor 1108a, act together to add/subtract the first doubleword of packed doubleword data Source2 (bit thirty-one through bit zero) to/from the first doubleword of packed word data Source1 (bit thirty-one through bit zero); adder/subtractors 908e-h, shown as virtual adder/subtractor 1108b, act together to add/subtract the second doubleword of packed doubleword data Source2 (bit sixty-three through bit thirty-two) to/from the second doubleword of packed doubleword data Source1 (bit sixty-three through bit thirty-two).

Virtual adder/subtractors 1108a-b communicate result information, via result out 1107a-b (combined result outs 907a-d and 907e-h), to virtual result registers 1110a-b. Each virtual result register 1110a-b (combined result registers 910a-d and 910e-h) stores a thirty-two bit result data element to be communicated onto Result bus 909.

Circuit Descriptions

The convention followed in the subsequent descriptions of circuits is that the bus names correspond to the signal names on that bus. For example, a Source1 signal is on a Source1 bus. Busses with multiple bits may be designated with particular bit ranges. For example, Source1<31:16> indicates that the bus corresponds to bits 31 through 16 of the Source1 bus. The whole bus may be referred to as the Source1 bus or Source1<63:0> (for a 64 bit bus). The complement of a signal may be referred to by appending an "#" after the signal name. For example, the complement of the Source1 signal on the Source1 bus is a Source1# signal on a Source1# bus.

Packed Adder/Subtractor Circuit

One embodiment of a packed adder/subtractor circuit is capable of performing a packed addition or packed subtraction on packed data elements which each contain two 32-bit values, four 16-bit data values, or eight 8-bit values. The packed adder/subtractor circuit adds three pairs of "dummy" bits between each byte of the two 32-bit data elements along with the 32-bit packed data elements to produce a 32-bit sum. A carry-input is used to add one during subtractions according to well-known methods. The three pairs of dummy bits between each byte are used to control the carry between each byte of the packed data elements. The sum of each pair of dummy bits is ignored in producing the 32-bit result. When a pair of these bits corresponds to a boundary between values in the packed data elements, zeroes or ones are generated depending on whether an add or subtract is to be performed and whether the packed data element is Source1 or Source2. Zero inputs on a pair of dummy bits (the corresponding bits of Source1 and Source2) disable the carry from the lower byte to the higher order byte according to two's complement arithmetic that is well known in the art. This is used during addition at the boundaries of values of Source1 and Source2. One inputs on a pair of dummy bits add one to the complement of the higher order bytes of Source2 to produce the two's complement of the higher order bytes at each

boundary between values of Source2 during subtraction operations according to two's complement arithmetic that is well known in the art. Since dummy bits do not control the carry-input of the lowest order byte of Source1 and Source2, the 35-bit adder with carry-in adds one to the lowest order byte during subtraction operations according to well-known methods. When one of these dummy bits do not correspond to a boundary between values in the packed data element, the highest order bit from the lower order byte is copied into this dummy bit. This forces the carry from the lower order byte to propagate into the carry input of the higher order byte according to two's complement arithmetic that is well known in the art. In another embodiment, a carry from the lower order byte is forced into the carry input of the higher order byte by inputting a one into one of the pair of dummy bits and a zero into the other one of the pair of dummy bits according to two's complement arithmetic that is well known in the art. Since the selection of the value for each dummy bit is performed in parallel with the selection of Source2 or its complement depending on whether a subtraction is performed, no additional time is required over a standard adder/subtractor circuit.

FIG. 12 illustrates one embodiment of an adder/subtractor circuit that is capable of performing additions and subtractions on one 32-bit data element, two 16-bit data elements, or eight 8-bit data elements. In order to perform operations on 64-bit data elements which can contain two 32-bit values, four 16-bit data values, or eight 8-bit values, two of the adder/subtractor circuits illustrated in FIG. 12 may be used, one for the most-significant 32-bits of the packed data element, and one for the least-significant 32-bit of the packed data element. A pair of dummy bits for controlling the carry from the least significant 32-bit data element into the most significant 32-bit data element may be used for 64-bit scalar additions and/or subtractions.

A Source1 bus 1203 and a Source2 bus 1204 each contain 32 bits of data that may be in one of several formats. The Source1 bus 1203 is coupled to logic 1200 which generates a 35-bit value (having the dummy bits embedded in the appropriate positions) on a A bus 1218. The Source2 bus

1204 is coupled to logic 1201 which generates a 35-bit value (having the dummy bits embedded in the appropriate positions) on a B bus 1219. A byte-indicator bus 1205 is driven high when Source1 and Source2 are packed byte data and low if Source1 and Source2 are packed word data or packed dword data. A word-indicator bus 1206 is driven high when Source1 and Source2 are packed word data and low if Source1 and Source2 are packed byte data or packed dword data. The byte-indicator bus 1205 is coupled to the input of inverter 1212 which drives byte-indicator# bus 1214. The byte-indicator bus 1205 and the word-indicator bus 1206 are coupled to the inputs of OR gate 1209 which drives byte-or-word indicator bus 1211. The byte or word indicator bus 1211 is coupled to the input of inverter 1213 which drives byte or word indicator# bus 1215. A subtract bus 1207 is driven high if the operation is a subtract. The byte-indicator bus 1205, the byte-indicator# bus 1214, the byte-or-word-indicator bus 1211, the byte-or-word-indicator# bus 1215, and the subtract bus 1207 are coupled to logic 1200. A compgt bus 1208 is driven high if the operation is a compare greater-than (compgt) operation. The subtract bus 1207 and the compgt bus 1208 are coupled to the inputs of OR gate 1210 which drives the subtract-or-compgt bus 1216. The byte-indicator# bus 1214, the byte-or-word-indicator# bus 1215, and the subtract-or-compgt bus 1216 are coupled to logic 1201.

The subtract bus 1207, the compgt bus 1208, the A bus 1218, and the B bus 1219 are coupled to a 35-bit adder 1202. In one embodiment, the 35-bit adder 1202 has a carry-in input that is coupled to the subtract bus 1207 to add 1 to the least significant value of the packed data for subtract operations by driving a one into the carry-in input of the 35-bit adder 1202 according to well-known methods. In one embodiment, the signals on the subtract bus 1207, the compgt bus 1208, .the byte-indicator bus 1205 and the word-indicator bus 1206 are decoded from a control signal 207 (referring to FIG. 2).

The (35-bit) A bus 1218, the (35-bit) B bus 1219, and the subtract bus 1207 are coupled to the inputs of the 35-bit adder 1202 which drives a (32-bit) result bus 1220. After an addition operation is performed, each value of the

result is the sum of the corresponding values of Source1 and Source2. After a subtraction operation is performed, each value of the result is the difference of the corresponding values of Source1 and Source2. After a compgt operation is performed, each value of the result is all ones when the corresponding value of Source1 is greater than the corresponding value of Source2 and all zeroes otherwise. The result bus 1220 does not include the bits corresponding to the sum of the dummy bits. The dummy bits are simply used to control the carry between the bytes depending on whether their is a boundary between packed data values corresponding to that bit and whether the operation is an addition or subtraction. The subtract operation controls whether the lowest order value of B should be incremented by one to adjust the complement of Source2 to be the two's complement of Source2 for subtraction operations. The dummy bits are positioned to add one to all of the values in the packed data for subtractions except for the lowest order value.

In one embodiment, the 35-bit adder 1202 includes saturation control logic. The saturation control logic overrides the sum with other values driven onto the result bus 1220, depending on the value of the sum. When there is an overflow condition for a value of the sum, the corresponding value of the result is the maximum value representable. When there is an underflow condition for a value of the sum, the corresponding value of the result is the minimum value representable. When neither an overflow or underflow condition exists for a value of the sum, the corresponding value of the result is the sum of the addition performed by the 35-bit adder 1202. In other embodiments, other conditions and alternative values may be used.

In one embodiment, the 35-bit adder 1202 is capable of performing a packed compare operation by forcing each value of the packed result to all ones or all zeroes depending on the result of a comparison of Source1 and Source2. In one embodiment, Source1 - Source2 + 1 is computed by complementing the Source2 input and performing an addition as described above. For each value in the packed data, the compare result is computed as follows:

compare = sign bit XNOR (carry-out XOR carry-in)

"Sign bit" indicates the sign bit (the most-significant bit in two's complement format) of the corresponding value of the packed data element. "Carry-out" indicates the carry-output of the most-significant bit of the corresponding value of the packed data element. "Carry-in" indicates the carry-input of the least-significant bit of the corresponding value of the packed data element. When the compare result is one, the corresponding value of the packed result is all ones. When the compare result is zero, the corresponding value of the packed result is all zeroes.

Figure 13 illustrates one embodiment of Logic 1200. Source1[31:24], Source1[23:16], Source1[15:8], Source1[7:0] of the Source1 bus 1203 are coupled to A[34:27], A[25:18], A[16:9], and A[7:0] of the A bus 1218, respectively. Each of these groups of data correspond to the smallest values represented in the packed data format (in this embodiment, a byte). When Source1 and Source2 are packed byte data, the values are added independently by forcing the dummy bits to have values that disable carry propagation between values of the packed data. When Source1 and Source2 are packed word data, the dummy bit corresponding to A[17] is selected (along with the dummy bit corresponding to B[17] described below) such that carries do not propagate from the lower order adjacent word to the higher order adjacent word and A[8] and A[26] are selected (along with the dummy bits corresponding to B[8] and B[26] described below) such that carries propagate from the least significant byte of the corresponding word to the most-significant byte of the corresponding word. When Source1 and Source2 are packed dword data, the dummy bits corresponding to A[8], A[17], and A[26] are selected (along with the dummy bits corresponding to B[8], B[17], and B[26] described below) such that carries propagate through all bytes of the dword.

A zero is input to the 0 inputs of mux1302 through zero busses 1310-1312, respectively. A one is input to the 1 inputs of mux1302 through one busses 1320-1322, respectively. Source1[23], Source1[15], and Source1[7] of the Source1 bus 1203 are coupled to the 2 inputs of mux1302, respectively.

The byte-indicator bus 1205 and the subtract bus 1217 are coupled to the inputs of AND gate 1306 which drives the subtract-byte bus 1303. The subtract-byte bus 1303 is coupled to the select zero (S0) inputs of the mux and the mux. The byte-indicator# bus 1214 is coupled to the select one (S1) input of the mux and the mux.

The mux and the mux correspond to a byte boundary of the packed data but do not correspond to a word or dword boundary. When the operation operates on packed byte data (the high order select input is zero), a zero input is selected if the operation is not a subtract (the low order select input is zero) and a one input is selected if the operation is a subtract (the low order select input is one). A zero input disables the carry between the two adjacent values of the packed data so that a packed addition, for example, of the bytes may be performed. A one input forces a one to be added to the higher order value of the adjacent values so that a packed subtraction of the bytes may be performed. The addition of one to the complement of each value of Source2 (described below) produces the two's complement of each value of Source2 except the lowest order value. A one is added to the lowest order value by the adder according to well-known methods for scalar adders since dummy bits are not used to control the carry-input of the lowest order value. Adding the two's complement of Source2 to Source1 produces the same result as subtracting Source2 from Source1. When the operation does not operate on packed byte data (the high order select input is one), the most-significant bit of the lower order adjacent value is selected. This propagates the carry from the lower order adjacent byte to the higher order adjacent byte.

The byte-or-word-indicator bus 1211 and the subtract bus 1217 are coupled to the inputs of AND gate 1307 which drives the subtract-byte-or-word bus 1304. The subtract-byte-or-word bus 1304 is coupled to the select

zero (S0) inputs of the mux. The byte-or-word indicator# bus 1215 is coupled to the select one (S1) input of the mux.

The mux correspond to a byte boundary and a word boundary of the packed data but do not correspond to a dword boundary. When the operation operates on packed byte data or packed word data (the high order select input is zero), a zero input is selected if the operation is not a subtract (the low order select input is zero) and a one input is selected if the operation is a subtract (the low order select input is one). A zero input disables the carry between the two adjacent independent values of the packed data so that a packed addition, for example, of the bytes may be performed. A one input forces a one to be added to an the more significant of the adjacent values so that a packed subtraction of the bytes or words may be performed. When the operation does not operate on packed byte or packed word data (the high order select input is one), the most-significant bit of the next lower order adjacent value is selected. This forces the carry from the next lower order byte to the next higher order byte.

Mux1302 produce A[26], A[17], and A[8] (the dummy bits) of the A bus 1218, respectively.

Figure 14 illustrates one embodiment of Logic 1201. Source2[31:24], Source2[23:16], Source2[15:8], Source2[7:0] of the Source2 bus 1204 are coupled to B[34:27], B[25:18], B[16:9], and B[7:0] of the B bus 1219, respectively. Each of these groups of data correspond to the smallest values represented in the packed data format.

A zero is input to the 0 inputs of mux1402 through zero busses 1410-1412, respectively. A one is input to the 1 inputs of mux1402 through zero busses 1420-1422, respectively. Source2[25], Source2[16], and Source2[7] of the Source2 bus 1204 are coupled to the 2 inputs of mux1402, respectively. Source2[25] of the Source2 bus 1204 is coupled to an inverter 1440 which drives the Source2# [25] bus 1430. Source2[16] of the Source2 bus 1204 is coupled to an inverter 1441 which drives the Source2#[16] bus 1431. Source2[7] of the Source2 bus 1204 is coupled to an inverter 1442 which drives the Source2[7]# bus 1432. The Source2#[25] bus 1430, Source2#[16]

bus 1431, and Source2#[7] bus 1432 are coupled to the 3 inputs of mux1402, respectively.

The subtract-or-compgt bus 1216 is coupled to the select zero (S0) inputs of mux and mux. The byte-indicator# bus 1214 is coupled to the select one (S1) input of the mux and the mux.

The mux and the mux correspond to a byte boundary of the packed data but do not correspond to a word or dword boundary. When the operation operates on packed byte data (the high order select input is zero), a zero input is selected if the operation is not a subtract or a compare greater-than operation (the low order select input is zero) and a one input is selected if the operation is a subtract or a compare greater-than operation (the low order select input is one). Zero inputs on a pair of dummy bits disables the carry between the two adjacent values of the packed data so that a packed addition, for example, of the bytes may be performed. A one input forces a one to be added to the higher order adjacent value so that a packed subtraction of the bytes may be performed. The addition of one to the complement of each value of Source2 (except the lowest order value) produces the two's complement of each value of Source2 (except the lowest order value). A one is added to the lowest order value by the 35-bit adder 1202 according to well-known methods for scalar adders. Adding the two's complement of Source2 to Source1 produces the same result as subtracting Source2 from Source1. When the operation does not operate on packed byte data (the high order select input is one), the most-significant bit of the adjacent value is selected if the operation is not a subtract or a compare greater-than operation (the low order select input is zero) and the complement of the most-significant bit of the adjacent value is selected if the operation is a subtract or a compare greater-than operation (the low order select input is one). This forces the carry from the lower order adjacent byte to the higher order adjacent byte.

The subtract-or-compgt bus 1216 is coupled to the select zero (S0) inputs of mux. The byte-or-word indicator# bus 1215 is coupled to the select one (S1) input of the mux.

The mux correspond to a byte boundary and a word boundary of the packed data but do not correspond to a dword boundary. When the operation operates on packed byte data or packed word data (the high order select input is zero), a zero input is selected if the operation is not a subtract or a compare greater-than operation (the low order select input is zero) and a one input is selected if the operation is a subtract or a compare greater-than operation (the low order select input is one). A zero input disables the carry between the two adjacent values of the packed data so that a packed addition, for example, of the bytes may be performed. A one input forces a one to be added to the higher order adjacent value so that a packed subtraction of the bytes or words may be performed. When the operation does not operate on packed byte or word data (the high order select input is one), the most-significant bit of the adjacent value is selected if the operation is not a subtract or a compare greater-than operation (the low order select input is zero) and the complement of the most-significant bit of the adjacent value is selected if the operation is a subtract or a compare greater-than operation (the low order select input is one). This forces the carry from the lower order adjacent byte to the higher order adjacent byte.

Mux1402 produce B[26], B[17], and B[8] (the dummy bits) of the B bus 1219, respectively.

While FIGS. 12, 13, and 14 illustrate one implementation of a packed adder and/or packed subtractor, any number of alternative packed adder and/or packed subtractor circuits could be used.

Adder/Subtractor Method

FIG. 15 illustrates one embodiment of a method to perform a packed add operation.

At Step 1501, a first packed data is accessed to from a register or another memory.

At Step 1502, a second packed data is accessed to from a register or another memory.

At Step 1503, the packed data element size is determined. The packed data element size may be a byte, word, or doubleword, for example. In one embodiment, the packed data element size is determined based on the control signal 207 (referring to FIG. 2).

At Step 1504, the muxes are driven to control the carry propagation based on the packed data element size determined in Step 1503 according to the methods described above.

At Step 1505, the first packed data and the second packed data are added to produce a packed data result.

FIG. 16 illustrates one embodiment of a method to perform a packed subtract operation.

At Step 1601, a first packed data is accessed to from a register or another memory.

At Step 1602, a second packed data is accessed to from a register or another memory.

At Step 1603, the second packed data is complemented to produce a third packed data. The complement of a two's complement value of N is -N-1.

At Step 1604, the packed data element size is determined. The packed data element size may be a byte, word, or doubleword, for example. In one embodiment, the packed data element size is determined based on the control signal 207 (referring to FIG. 2).

At Step 1605, the muxes are driven to add one to each value (except the least significant value) of the third packed data based on the packed data element size determined in Step 1604 according to the methods described above. This operation negates all values (except the least significant value) of the third packed data. The complement plus one of a two's complement value of N is -N.

At Step 1606, the first packed data and the third packed data are added to produce a packed result. The carry-input is one to add one to the least-significant value of the third packed data. Each of the values of the packed

result is the corresponding value of the first packed data minus the corresponding value of the second packed data.

At Step 1701, a first packed data is accessed to from a register or another memory.

At Step 1702, a second packed data is accessed to from a register or another memory.

At Step 1703, the second packed data is complemented to produce a third packed data.

At Step 1704, the packed data element size is determined. The packed data element size may be a byte, word, or doubleword, for example. In one embodiment, the packed data element size is determined based on the control signal 207 (referring to FIG. 2).

At Step 1705, the muxes are driven to propogate the carry of the third packed data based on the packed data element size determined in Step 1604 according to the methods described above.

At Step 1706, the first packed data and the third packed data are added to produce a packed result. The carry-input is zero. Each of the values of the packed result is the corresponding value of the first packed data minus the corresponding value of the second packed data minus one.

At Step 1707, saturation control is performed to force values of the packed result to all ones or all zeroes depending on the packed result according to the methods described above.

Other Embodiments

Thus, an apparatus and method of increasing processor performance by efficiently reducing the difference between data element length and actual carrying and storage capacity has been described. In particular, a processor capable of performing addition and subtraction operations using packed data has been described.

CLAIMS

What is claimed is:

1. A microprocessor comprising an apparatus for performing a packed arithmetic operation, said apparatus comprising:

an adder having a plurality of first inputs, a plurality of second inputs, and a plurality of outputs, a packed result of said packed arithmetic operation being indicated on a portion of said plurality of outputs;

a first bus coupled to a plurality of first portions of said first inputs corresponding to a first plurality of values, a first packed data representing said first plurality of values being driven on said first bus;

a second bus coupled to a plurality of second portions of said second inputs corresponding to a second plurality of values, a second packed data representing said second plurality of values being driven on said second bus;

at least one first mux coupled to at least one third portion of said first inputs, each of said at least one third portion being coupled between two of said plurality of first portions; and

at least one second mux coupled to at least one fourth portion of said first inputs, each of said at least one fourth portion being coupled between two of said plurality of second portions, said at least one first mux and said at least one second mux controlling carry propagation between at least two of a third plurality of values in said adder, said packed result representing said third plurality of values;

2. The microprocessor of Claim 1 wherein said at least one first mux and at least one second mux controls whether a one is added to at least one of said third plurality of values.

3. The microprocessor of Claim 2 wherein said at least one first mux and said at least one second mux control whether one is added to at least one of said third plurality of values.

4. The microprocessor of Claim 1 wherein said apparatus further comprises a third input of each of said plurality of muxes being coupled to a logic zero.

5. The microprocessor of Claim 1 wherein said apparatus further comprises a third input of each of said plurality of muxes being coupled to a logic one.

6. A circuit comprising apparatus for performing a packed arithmetic operation, said circuit comprising:

an adder having a plurality of first inputs, a plurality of second inputs, and a plurality of outputs, a packed result of said packed arithmetic operation being indicated on a portion of said plurality of outputs;

a first bus coupled to a plurality of first portions of said first inputs corresponding to a first plurality of values, a first packed data representing said first plurality of values being driven on said first bus;

a second bus coupled to a plurality of second portions of said second inputs corresponding to a second plurality of values, a second packed data representing said second plurality of values being driven on said second bus;

at least one first mux coupled to at least one third portion of said first inputs, each of said at least one third portion being coupled between two of said plurality of first portions; and

at least one second mux coupled to at least one fourth portion of said

first inputs, each of said at least one fourth portion being coupled between two of said plurality of second portions, said at least one first mux and said at least one second mux controlling carry propagation between at least two of a third plurality of values in said adder, said packed result representing said third plurality of values;

7. The circuit of Claim 6 wherein said at least one second mux controls whether at least one of said second plurality of values is inverted.

8. The microprocessor of Claim 7 wherein said at least one first mux and said at least one second mux control whether one is added to at least one of said third plurality of values.

9. The microprocessor of Claim 6 wherein said apparatus further comprises a third input of each of said plurality of muxes being coupled to a logic zero.

10. The microprocessor of Claim 6 wherein said apparatus further comprises a third input of each of said plurality of muxes being coupled to a logic one.

11. The method of performing a packed addition, said method comprising the steps of:

accessing a first packed data from a memory;

accessing a second packed data from said memory;

determining the size of said first packed data elements and said second

packed data elements;

driving a plurality of muxes to control carry propagation based on the size of said first packed data elements and said second packed data elements; and

adding first packed data to second packed data to produce a packed result.

12. The method of performing a packed subtraction, said method comprising the steps of:

accessing a first packed data from a memory;

accessing a second packed data from said memory;

complementing said second packed data to produce a third packed data;

determining the size of said first packed data elements and said second packed data elements;

driving a plurality of muxes to add one to all but the least-significant of the values of the second packed data elements based on the size of said first packed data elements and said second packed data elements; and

adding first packed data to second packed data with a carry-in of one to produce a packed result.

13. The method of performing a packed compare greater-than, said method comprising the steps of:

accessing a first packed data from a memory;

accessing a second packed data from said memory;

complementing the second packed data to produce a third packed data;

determining the size of said first packed data elements and said second packed data elements;

driving a plurality muxes to control carry propagation based on the size of said first packed data elements and said second packed data elements;

adding first packed data to second packed data with a carry-in of zero

to produce a packed intermediate result having a plurality of values; and

performing saturation control to force values of to all ones or all zeroes based on said packed intermediate result to produce a packed result.

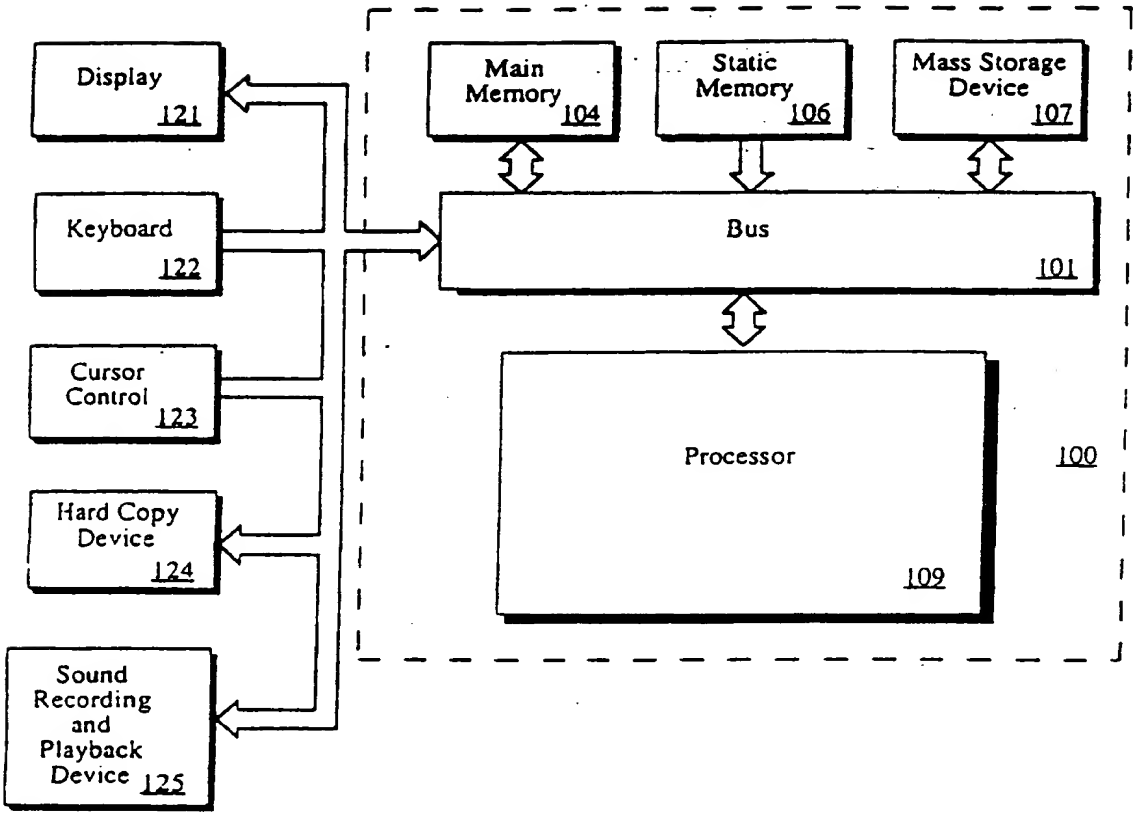

Display
121
Keyboard
122
Cursor
Control
123
Hard Copy
Device
124
Sound
Recording
and
Playback
Device 125
Main
Memory 104
Static
Memory
106
Mass Storage
Device
107
Bus
101
Processor
109
100

Figure 1

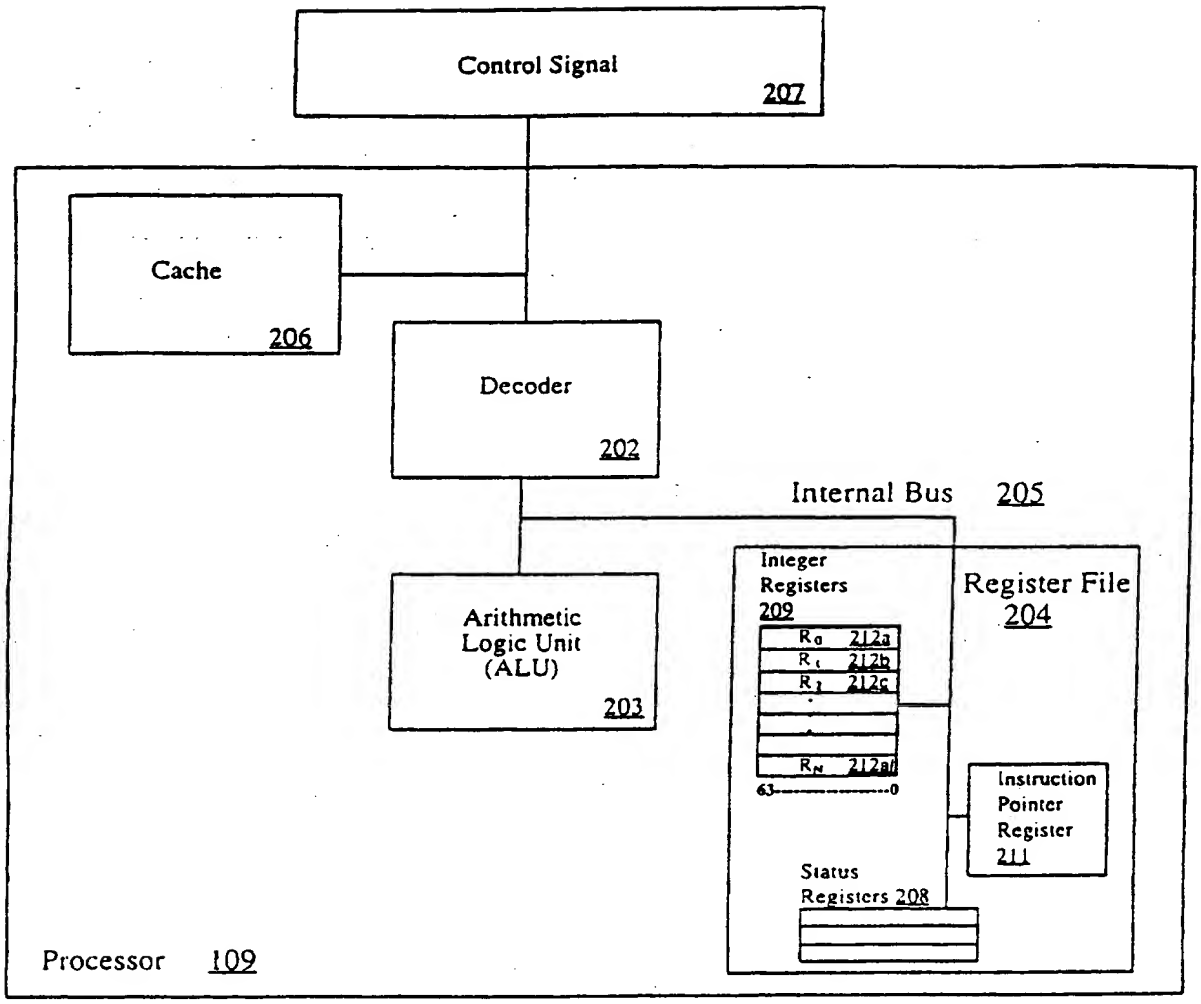
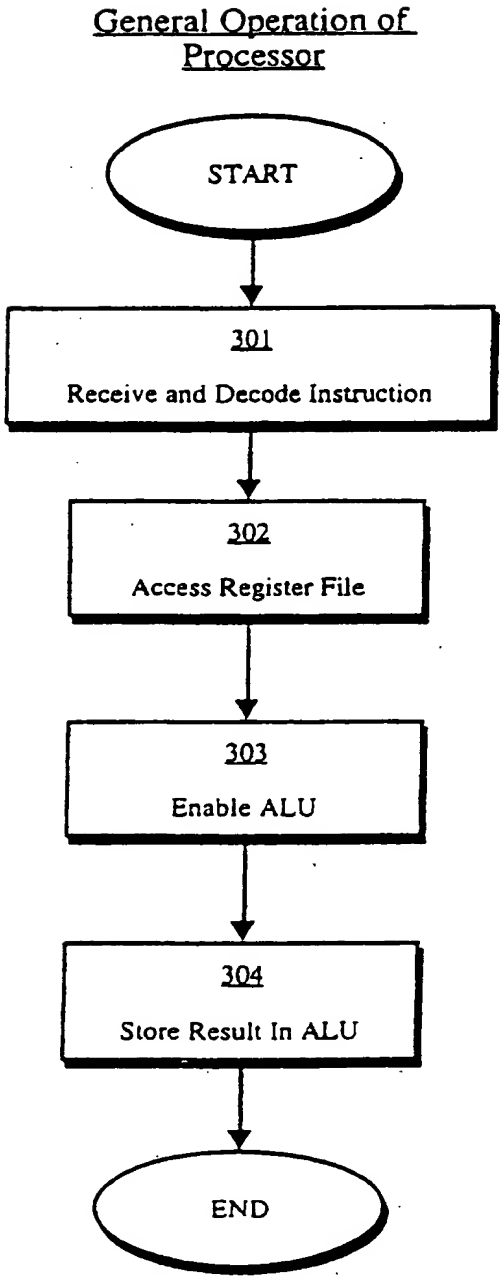

Control Signal
207
Cache
206
Decoder
202
Internal Bus 205
Arithmetic
Logic Unit
(ALU)
203
Integer
Registers
209
Register File
204
Instruction
Pointer
Register
211
Status
Registers 208
Processor 109

Figure 2

Figure 3


General Operation of Processor
START
301
Receive and Decode Instruction
302
Access Register File
303
Enable ALU
304
Store Result In ALU
END

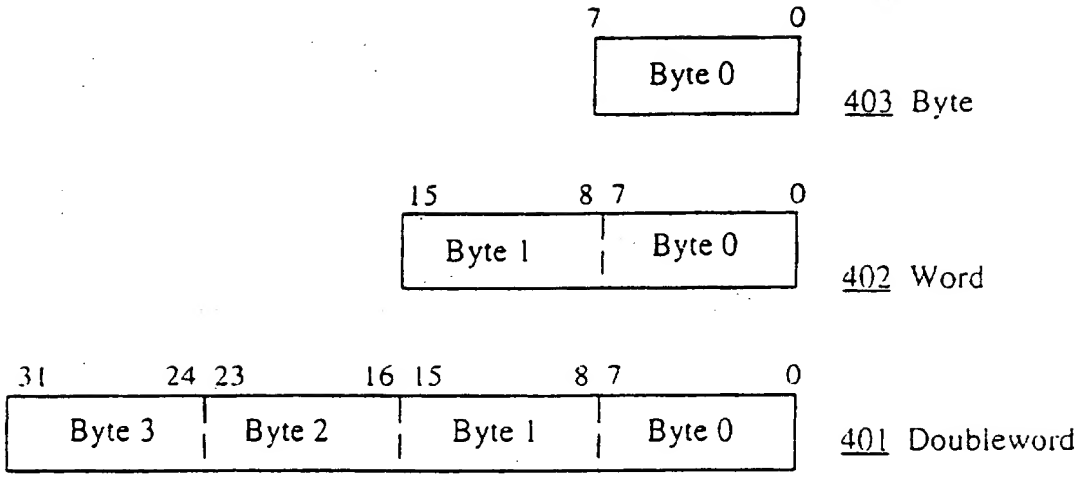

7
0
Byte 0
403 Byte
15
8 7
0
Byte 1
Byte 0
402 Word
31
24 23
16 15
8 7
0
Byte 3
Byte 2
Byte 1
Byte 0
401 Doubleword

Figure 4a

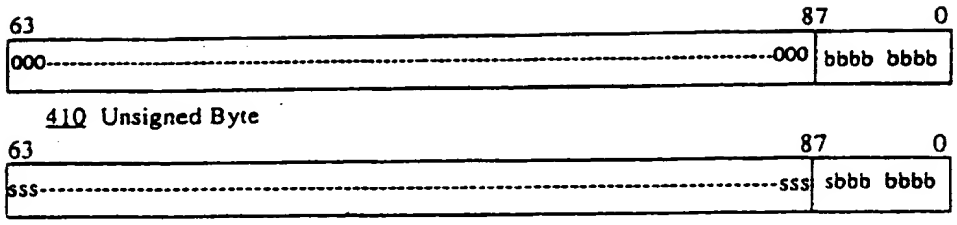


Figure 4b

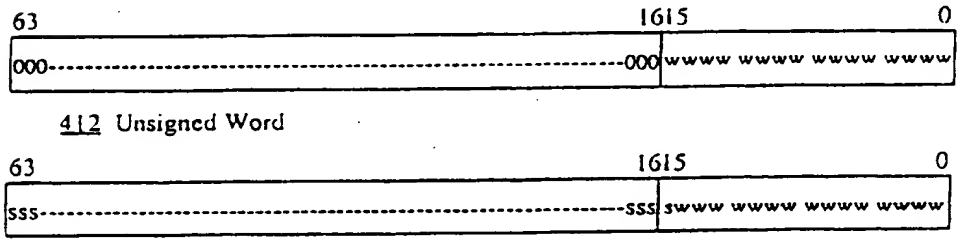


Figure 4c

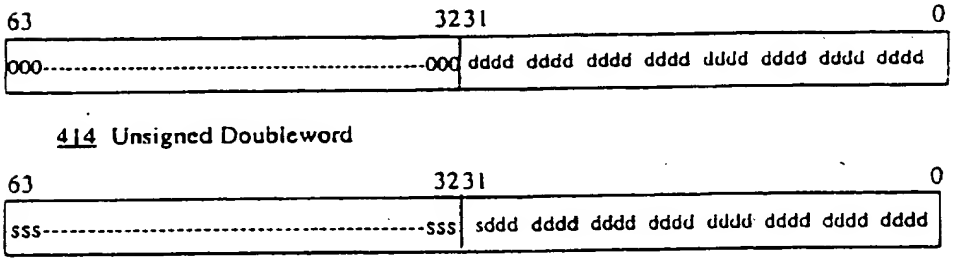
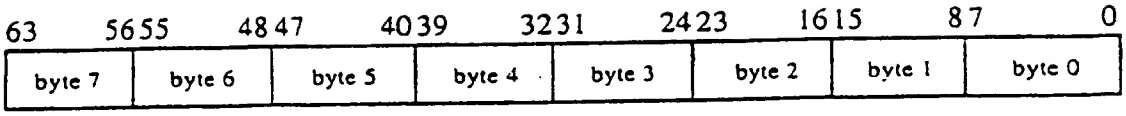
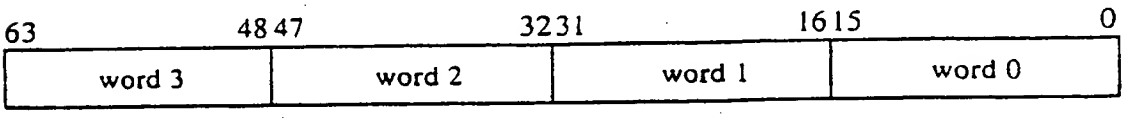


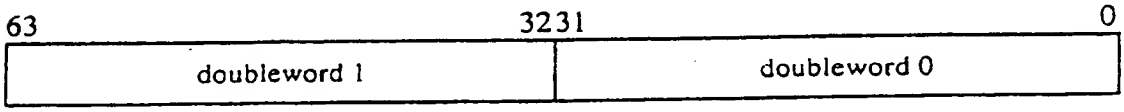
Figure 4d



501 Packed Byte Data Type.

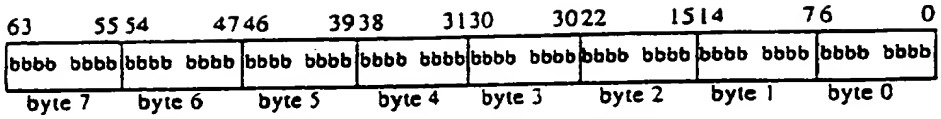


502 Packed Word Data Type.

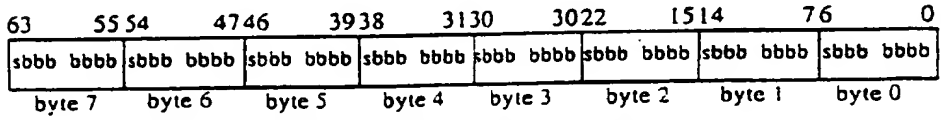


503 Packed Doubleword Data Type.

Figure 5a

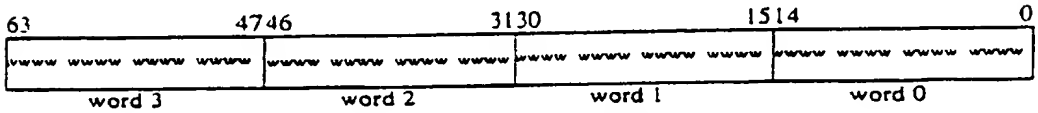


510 Unsigned Packed Byte In-Register Representation

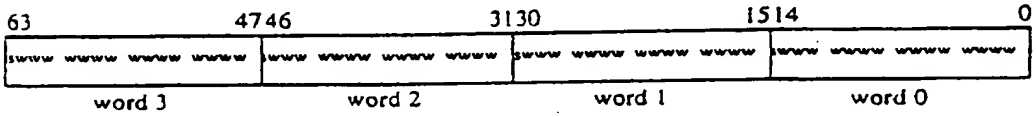


511 Signed Packed Byte In-Register Representation

Figure 5b

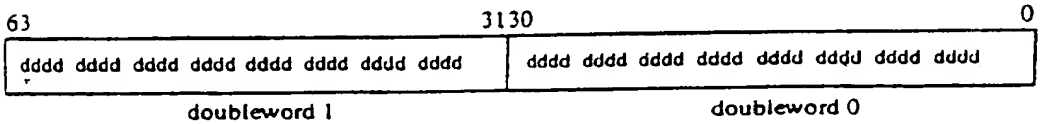


512 Unsigned Packed Word In-Register Representation

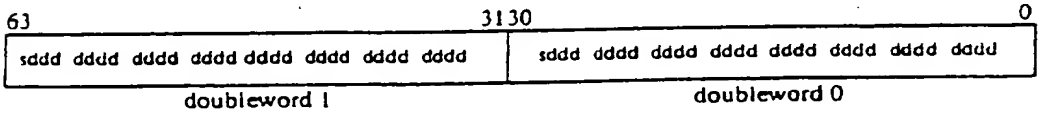


513 Signed Packed Word In-Register Representation

Figure 5c

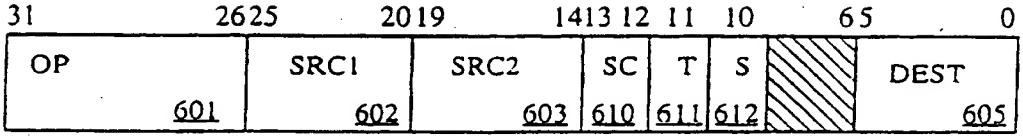


514 Unsigned Packed Doubleword In-Register Representation



515 Signed Packed Doubleword In-Register Representation

Figure 5d



207 Control Signal

Figure 6

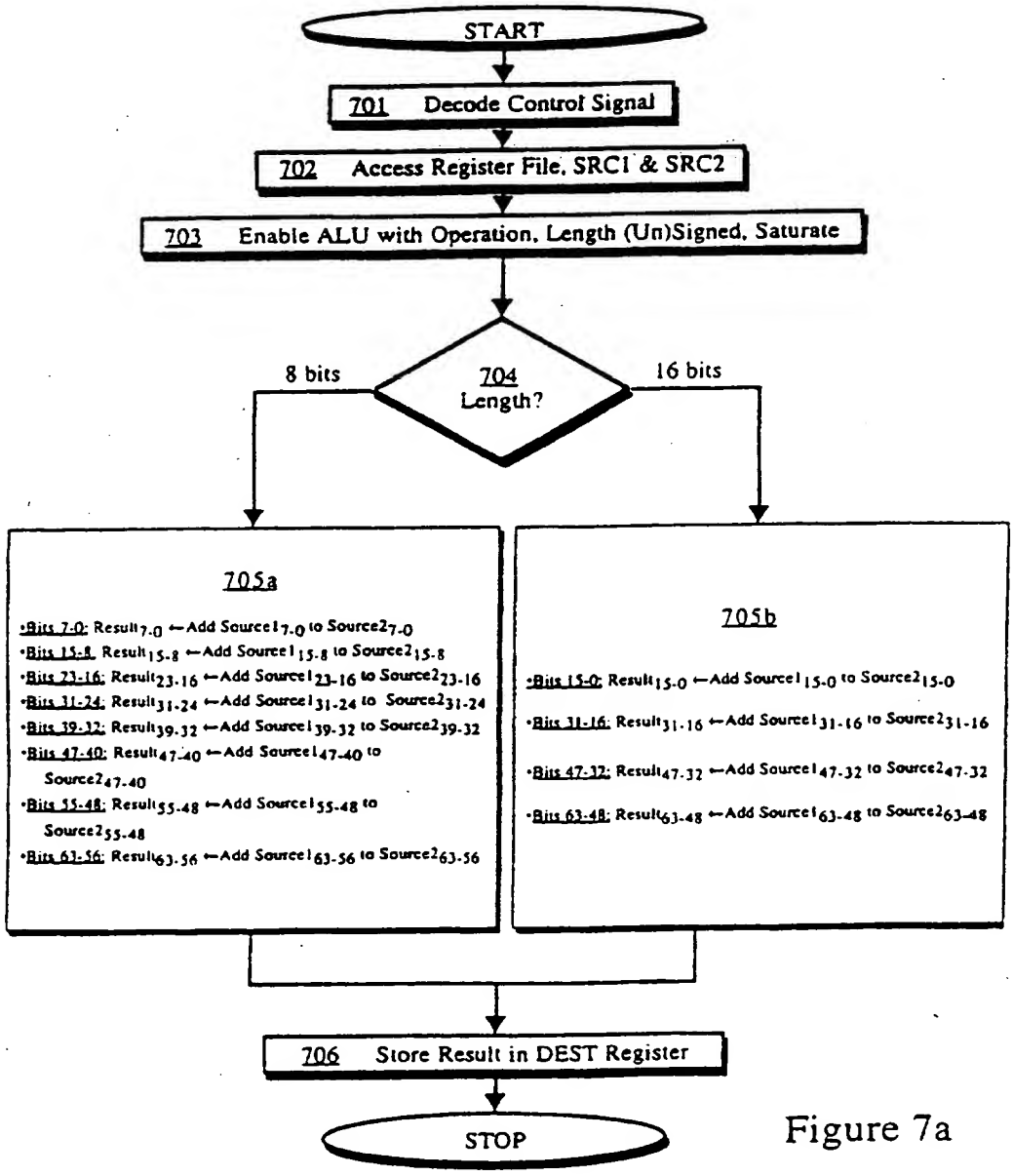

START
701 Decode Control Signal
702 Access Register File, SRC1 & SRC2
703 Enable ALU with Operation, Length (Un)Signed, Saturate
704 Length?
8 bits
16 bits
705a
•Bits 7-0: Result7-0 ←Add Source1 7-0 to Source2 7-0
•Bits 15-8: Result15-8 ←Add Source1 15-8 to Source2 15-8
•Bits 23-16: Result23-16 ←Add Source1 23-16 to Source2 23-16
•Bits 31-24: Result31-24 ←Add Source1 31-24 to Source2 31-24
•Bits 39-32: Result39-32 ←Add Source1 39-32 to Source2 39-32
•Bits 47-40: Result47-40 ←Add Source1 47-40 to Source2 47-40
•Bits 55-48: Result55-48 ←Add Source1 55-48 to Source2 55-48
•Bits 63-56: Result63-56 ←Add Source1 63-56 to Source2 63-56
705b
•Bits 15-0: Result15-0 ←Add Source1 15-0 to Source2 15-0
•Bits 31-16: Result31-16 ←Add Source1 31-16 to Source2 31-16
•Bits 47-32: Result47-32 ←Add Source1 47-32 to Source2 47-32
•Bits 63-48: Result63-48 ←Add Source1 63-48 to Source2 63-48
706 Store Result in DEST Register
STOP

Figure 7a

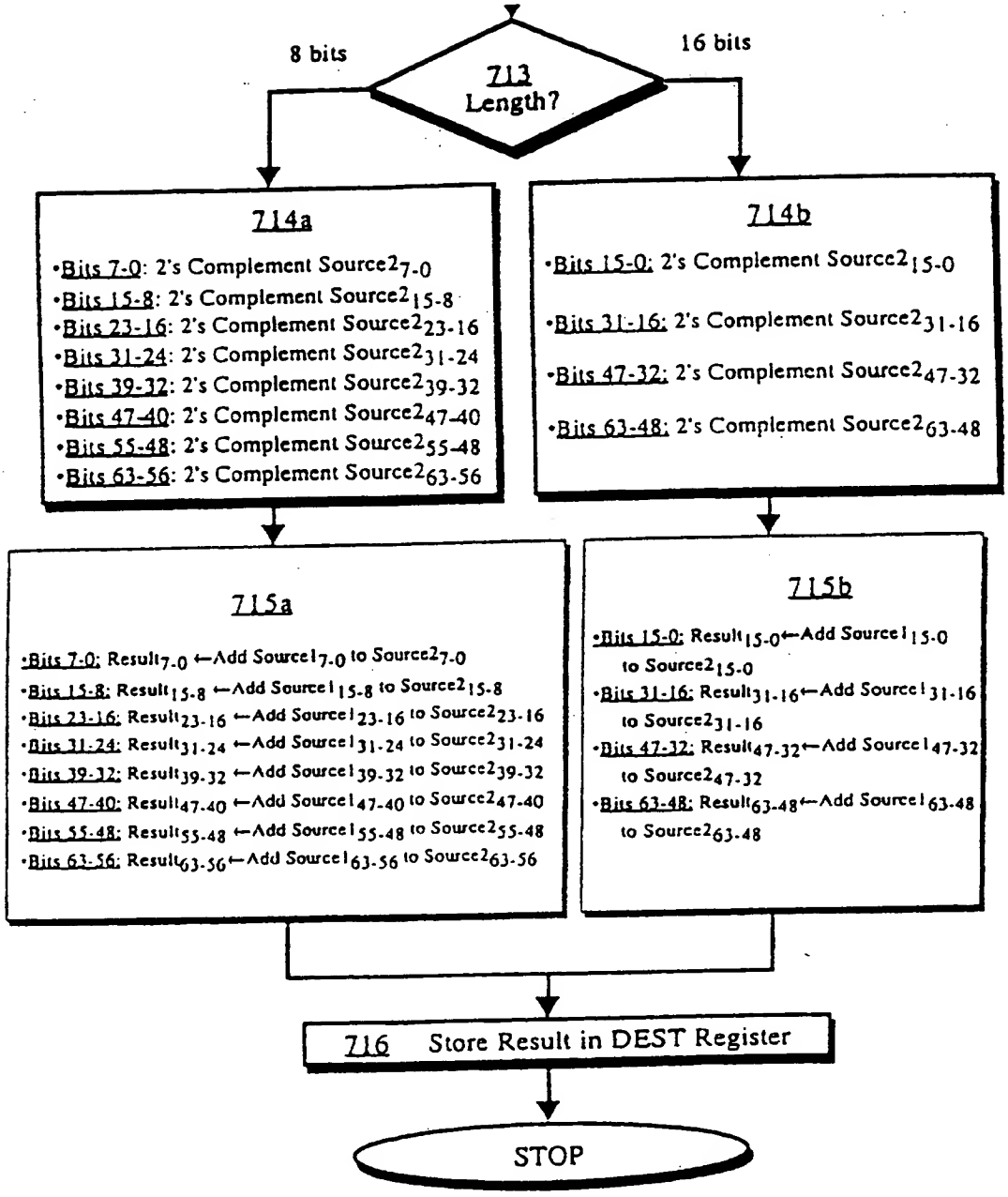

713
Length?
8 bits
16 bits
714a
•Bits 7-0: 2's Complement Source2 7-0
•Bits 15-8: 2's Complement Source2 15-8
•Bits 23-16: 2's Complement Source2 23-16
•Bits 31-24: 2's Complement Source2 31-24
•Bits 39-32: 2's Complement Source2 39-32
•Bits 47-40: 2's Complement Source2 47-40
•Bits 55-48: 2's Complement Source2 55-48
•Bits 63-56: 2's Complement Source2 63-56
714b
•Bits 15-0: 2's Complement Source2 15-0
•Bits 31-16: 2's Complement Source2 31-16
•Bits 47-32: 2's Complement Source2 47-32
•Bits 63-48: 2's Complement Source2 63-48
715a
•Bits 7-0: Result 7-0 ←Add Source1 7-0 to Source2 7-0
•Bits 15-8: Result 15-8 ←Add Source1 15-8 to Source2 15-8
•Bits 23-16: Result 23-16 ←Add Source1 23-16 to Source2 23-16
•Bits 31-24: Result 31-24 ←Add Source1 31-24 to Source2 31-24
•Bits 39-32: Result 39-32 ←Add Source1 39-32 to Source2 39-32
•Bits 47-40: Result 47-40 ←Add Source1 47-40 to Source2 47-40
•Bits 55-48: Result 55-48 ←Add Source1 55-48 to Source2 55-48
•Bits 63-56: Result 63-56 ←Add Source1 63-56 to Source2 63-56
715b
•Bits 15-0: Result 15-0 ←Add Source1 15-0 to Source2 15-0
•Bits 31-16: Result 31-16 ←Add Source1 31-16 to Source2 31-16
•Bits 47-32: Result 47-32 ←Add Source1 47-32 to Source2 47-32
•Bits 63-48: Result 63-48 ←Add Source1 63-48 to Source2 63-48
716 Store Result in DEST Register
STOP

Figure 7b

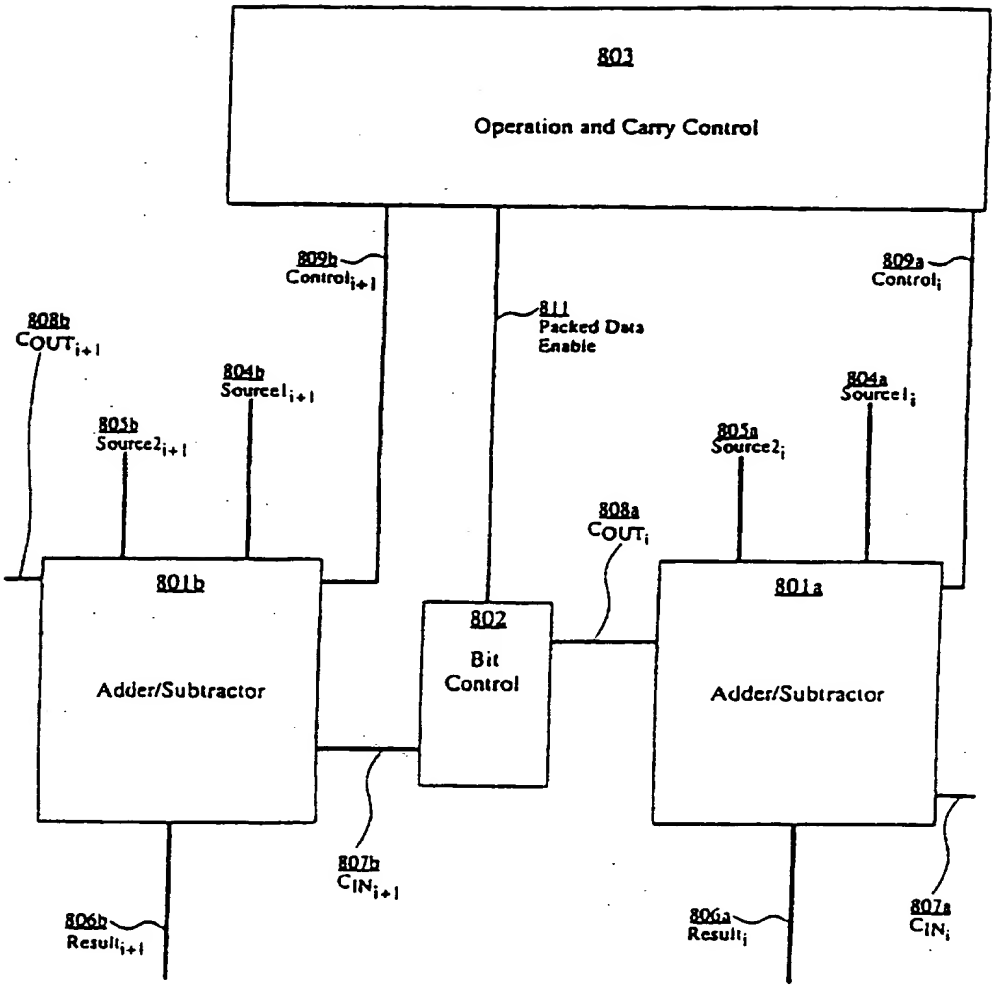

803
Operation and Carry Control
809b
Control$_{i+1}$
809a
Control$_i$
811
Packed Data
Enable
808b
COUT$_{i+1}$
804b
Source1$_{i+1}$
805b
Source2$_{i+1}$
804a
Source1$_i$
805a
Source2$_i$
808a
COUT$_i$
801b
Adder/Subtractor
802
Bit
Control
801a
Adder/Subtractor
807b
CIN$_{i+1}$
806b
Result$_{i+1}$
806a
Result$_i$
807a
CIN$_i$

Figure 8

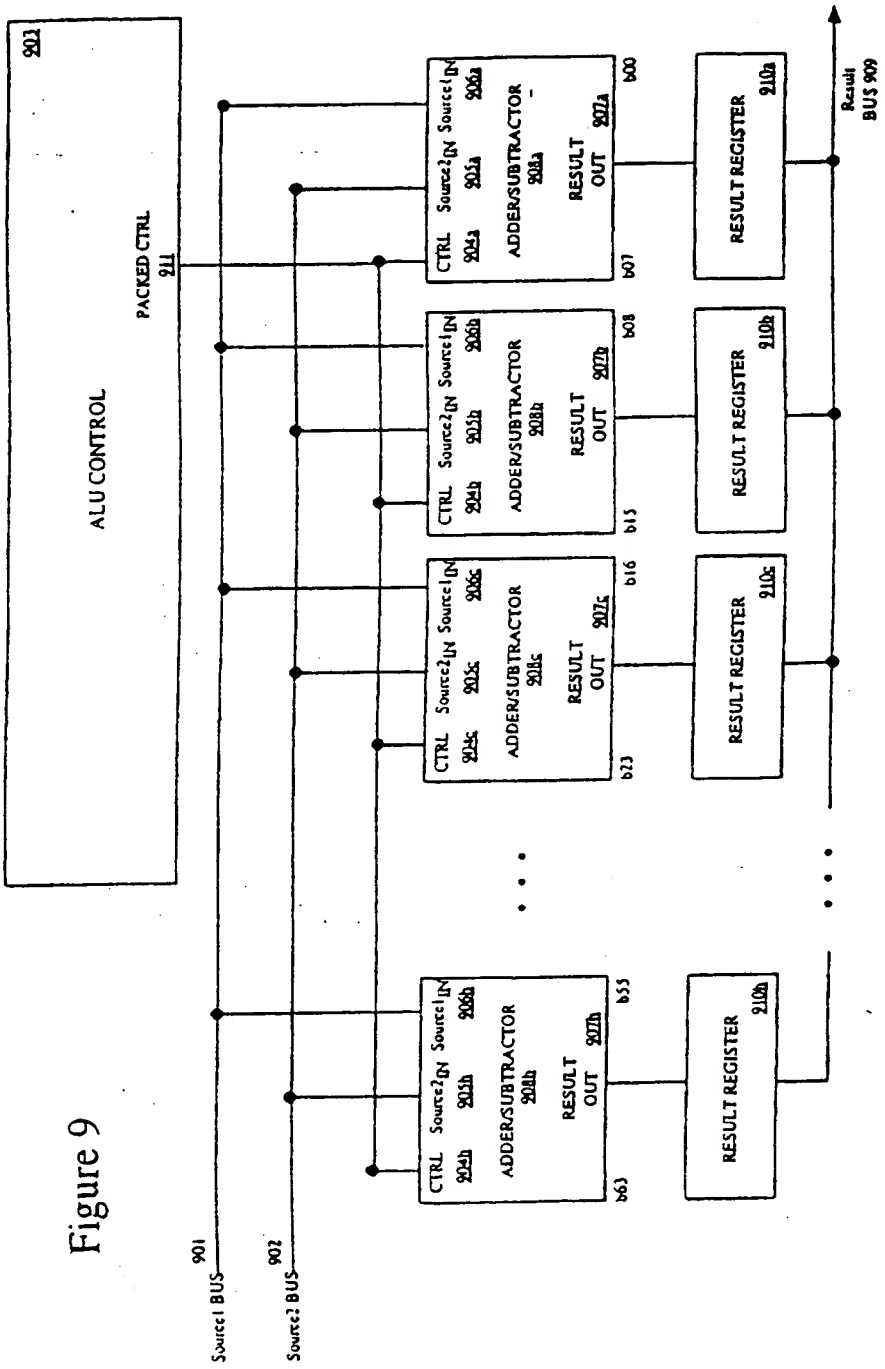

ALU CONTROL
PACKED CTRL
Source1 BUS
Source2 BUS
ADDER/SUBTRACTOR
RESULT REGISTER
Result BUS

Figure 9

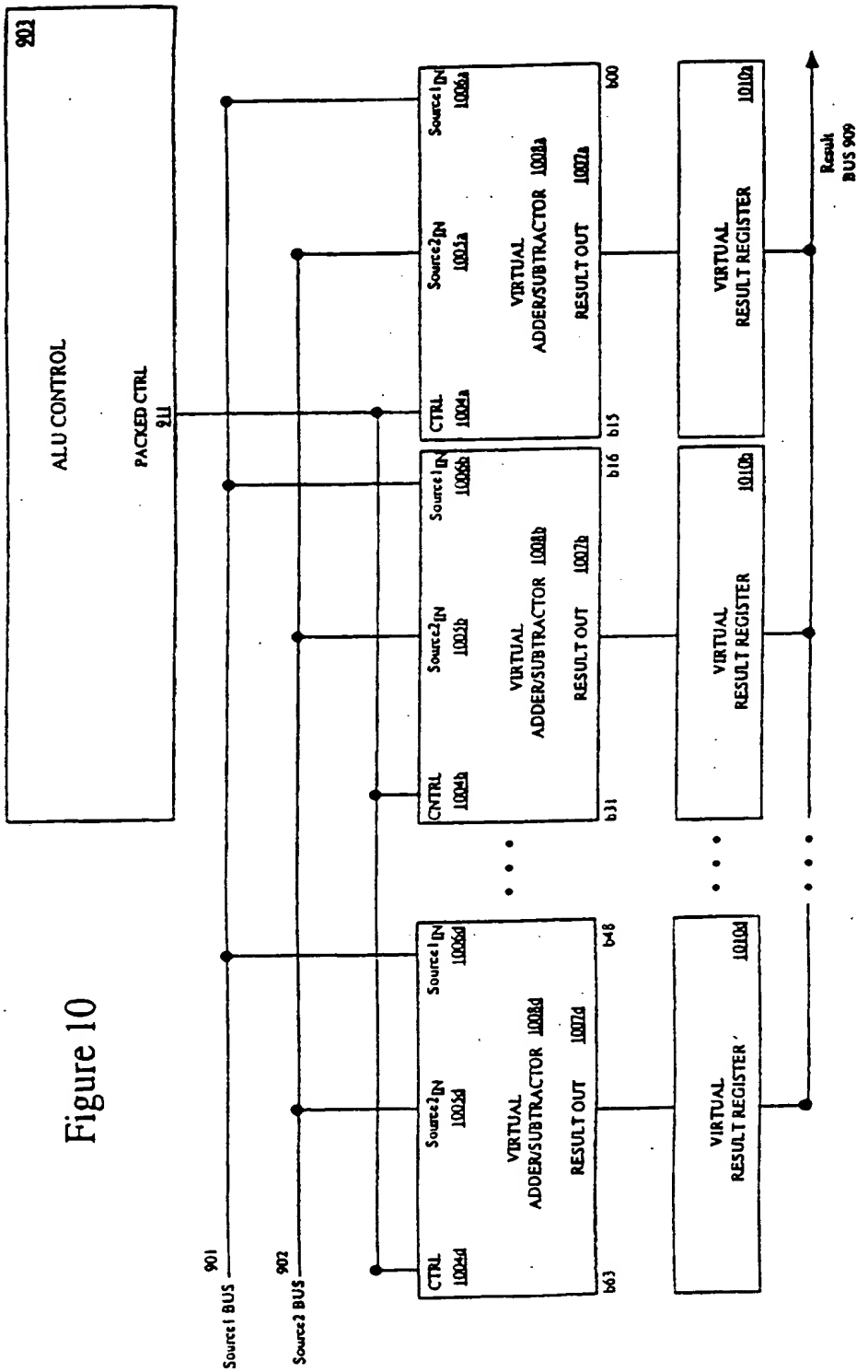

903
ALU CONTROL
PACKED CTRL
911
Source1 BUS
901
Source2 BUS
902
Source1 IN
Source2 IN
CTRL
VIRTUAL
ADDER/SUBTRACTOR
RESULT OUT
VIRTUAL
RESULT REGISTER
Result
BUS 909
b00
b15
b16
b31
b48
b63

Figure 10

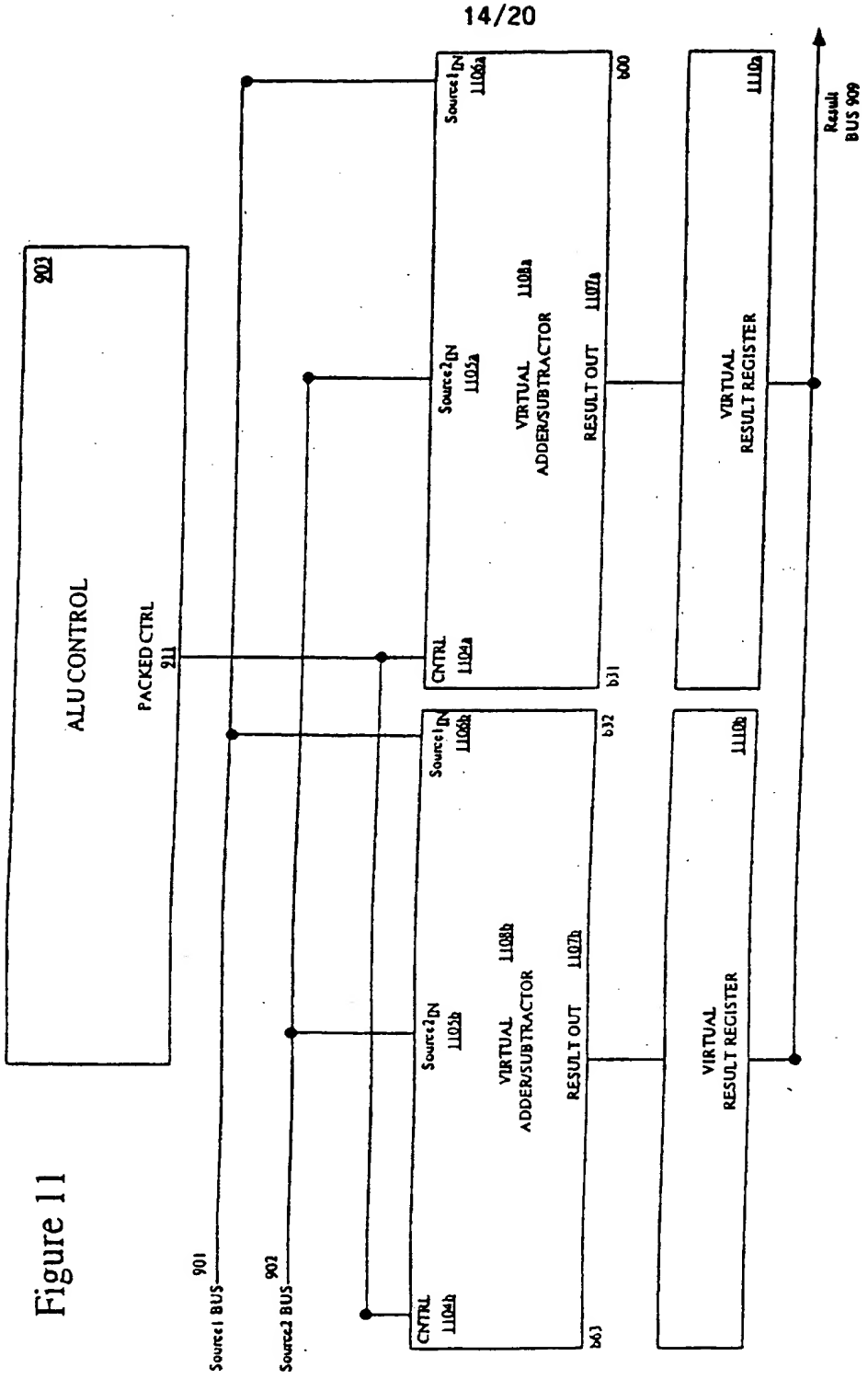

ALU CONTROL
PACKED CTRL
211
903
Source1 BUS
901
Source2 BUS
902
Source1 IN
1106a
Source2 IN
1102a
CNTRL
1104a
VIRTUAL
ADDER/SUBTRACTOR
1108a
RESULT OUT
1107a
b31
b00
VIRTUAL
RESULT REGISTER
1110a
Result
BUS 909
Source1 IN
1106b
Source2 IN
1102b
CNTRL
1104b
VIRTUAL
ADDER/SUBTRACTOR
1108b
RESULT OUT
1107b
b32
b63
VIRTUAL
RESULT REGISTER
1110b

Figure 11

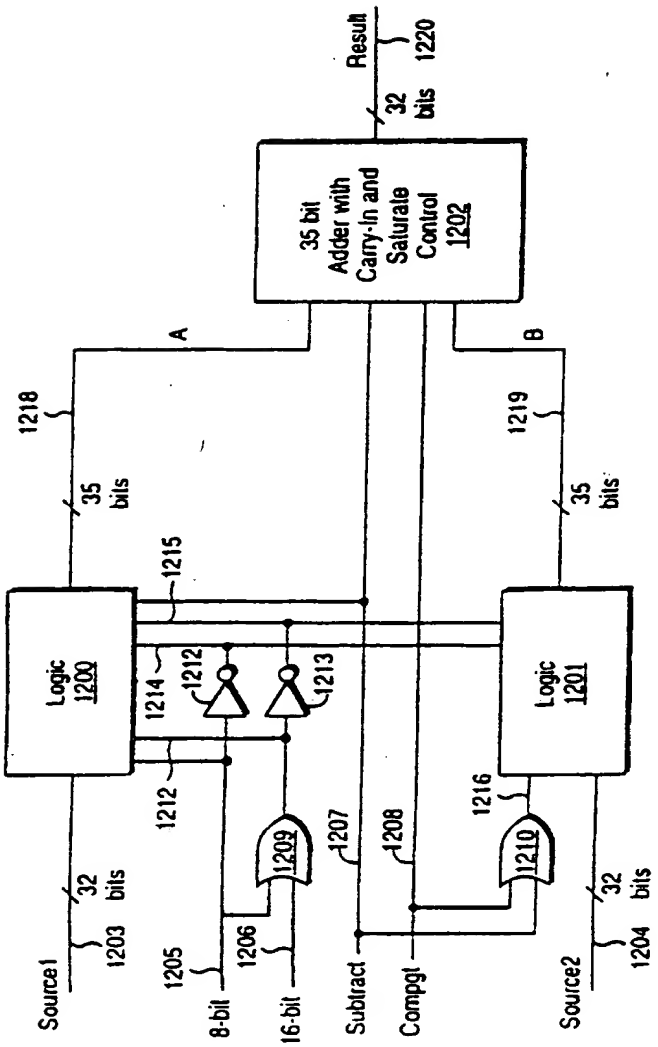

Result
1220
32 bits
35 bit Adder with Carry-In and Saturate Control 1202
A
B
1218
1219
35 bits
35 bits
1215
Logic 1200
Logic 1201
1214
1212
1213
1212
1207
1208
1216
1209
1210
32 bits
32 bits
1203
1204
Source1
Source2
1205
1206
8-bit
16-bit
Subtract
Compgt

FIG. 12

FIG. 13

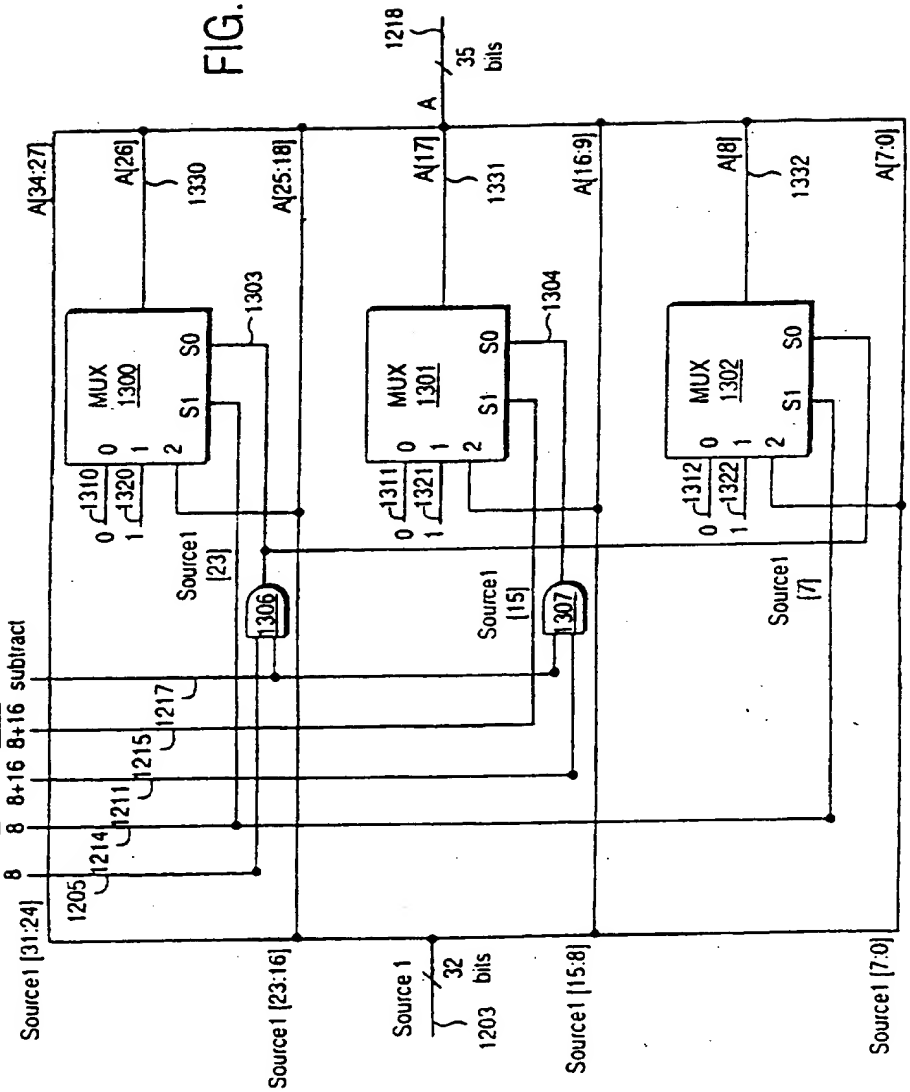
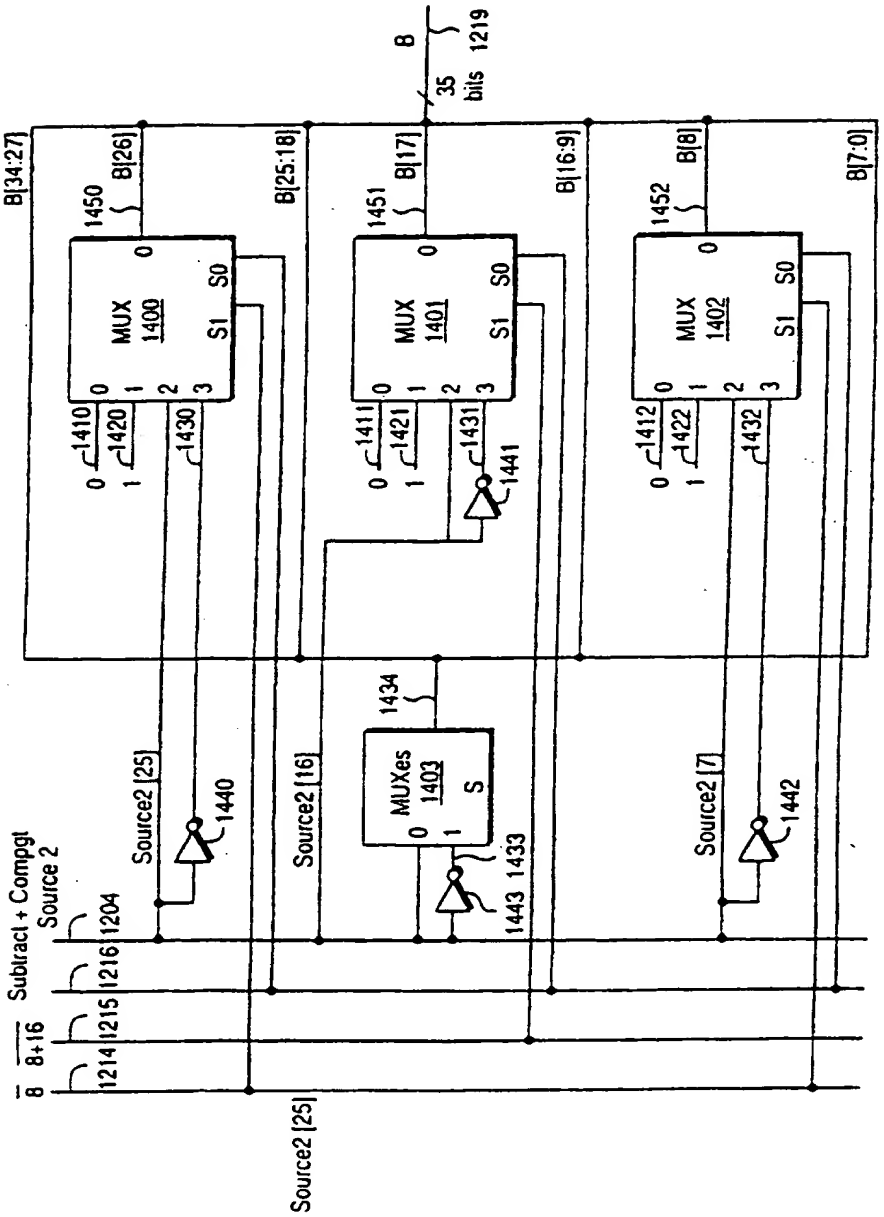

A[34:27]
A[26]
1330
A[25:18]
A[17]
1331
A[16:9]
A[8]
1332
A[7:0]
1218
A
35 bits
MUX 1300
MUX 1301
MUX 1302
S0
S1
1303
1304
1310
1320
1311
1321
1312
1322
Source1 [23]
Source1 [15]
Source1 [7]
1306
1307
subtract
8+16
8+16
8
8
1217
1215
1211
1214
1205
Source1 [31:24]
Source1 [23:16]
Source1
32 bits
1203
Source1 [15:8]
Source1 [7:0]

FIG. 14


B
35 bits
1219
B[34:27]
B[25:18]
B[16:9]
B[7:0]
1450
1451
1452
B[26]
B[17]
B[8]
MUX 1400
MUX 1401
MUX 1402
S0
S1
1410
1420
1430
1411
1421
1431
1441
1412
1422
1432
1434
MUXes 1403
S
1443
1433
1440
1442
Source2 [25]
Source2 [16]
Source2 [7]
Subtract + Compgt Source 2
1204
1216
1215
1214
8+16
8
Source2 [25]

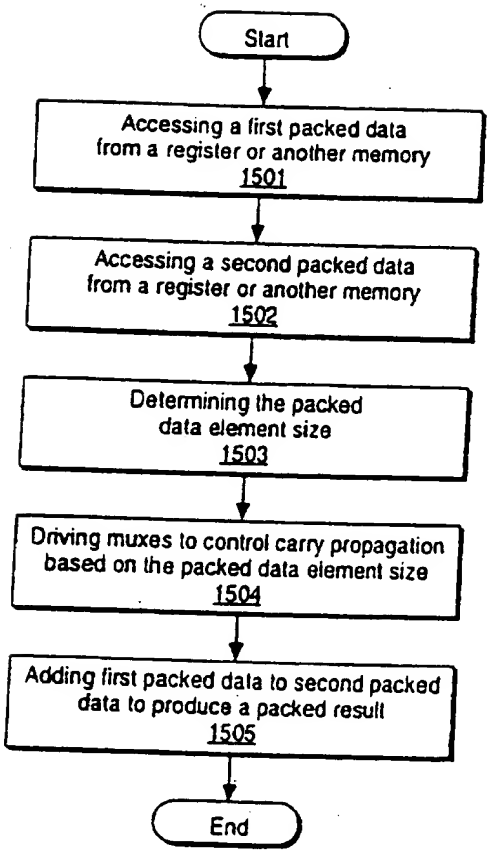

Start
Accessing a first packed data from a register or another memory
1501
Accessing a second packed data from a register or another memory
1502
Determining the packed data element size
1503
Driving muxes to control carry propagation based on the packed data element size
1504
Adding first packed data to second packed data to produce a packed result
1505
End

FIG. 15

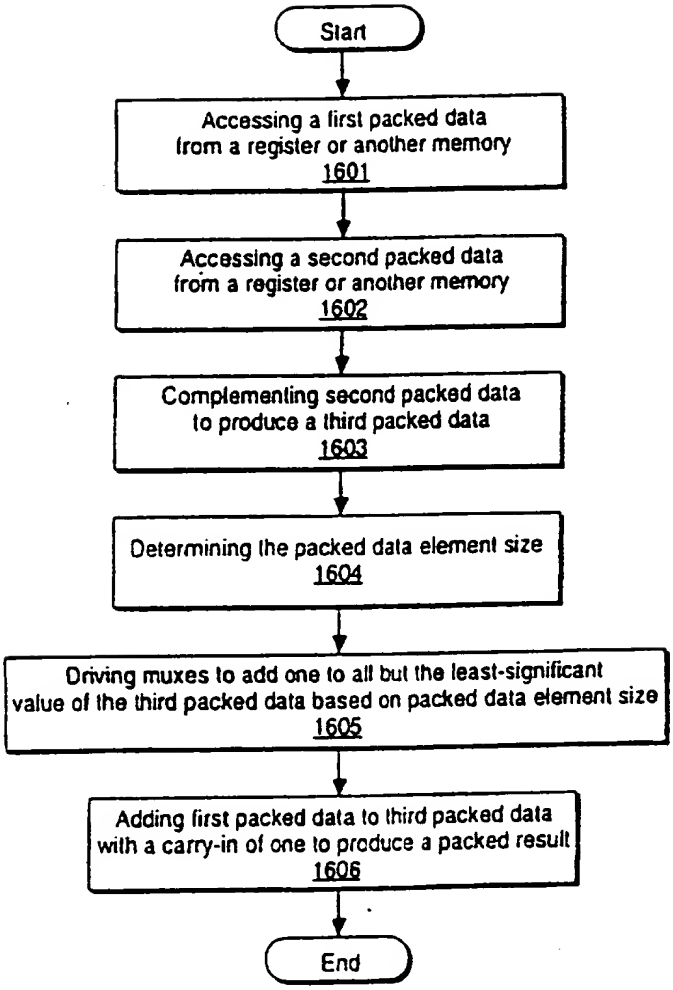

Start
Accessing a first packed data from a register or another memory 1601
Accessing a second packed data from a register or another memory 1602
Complementing second packed data to produce a third packed data 1603
Determining the packed data element size 1604
Driving muxes to add one to all but the least-significant value of the third packed data based on packed data element size 1605
Adding first packed data to third packed data with a carry-in of one to produce a packed result 1606
End

FIG. 16

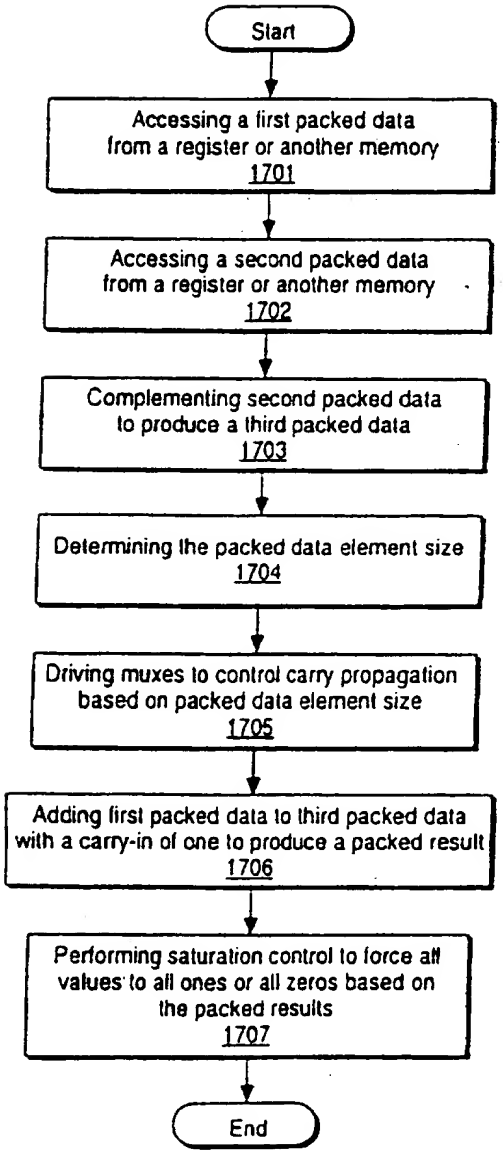

Start
Accessing a first packed data from a register or another memory 1701
Accessing a second packed data from a register or another memory 1702
Complementing second packed data to produce a third packed data 1703
Determining the packed data element size 1704
Driving muxes to control carry propagation based on packed data element size 1705
Adding first packed data to third packed data with a carry-in of one to produce a packed result 1706
Performing saturation control to force all values to all ones or all zeros based on the packed results 1707
End

FIG. 17

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/03445

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 7/38, 7/42, 7/50
US CL : 395/800, 421.07; 364/772, 721, 728.01, 736, 754, 748, 745
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/800, 421.07; 364/772, 721, 728.01, 736, 754, 748, 745

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

NONE

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS USPAT Database
APS JPOABS Database

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US, 3,986,015 A (GOODING et al.) 12 October 1976, col.6, lines 15-2.	1-13
X	US, 4,888,719 A (YASSA) 19 December 1989, col.4, lines 54-68; col.5, lines 1-3.	1,12
X	US, 5,189,635 A (OHKI) 23 February 1993, col.4, lines 21-41.	1-13
X	US, 4,821,225 A (ANDO et al.) 11 April 1989, col.2, lines 55-66.	1-13

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search
07 APRIL 1997

Date of mailing of the international search report
13 MAY 1997

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
DANIEL H. PAN
Telephone No. (703) 305-9600

Form PCT/ISA/210 (second sheet)(July 1992)*

THIS PAGE BLANK (USPTO)

THIS PAGE BLANK (USPTO)